PROGRAMACIÓN EN INTERNET

Clientes Web

Sergio Luján Mora



www.ecu.fm

PROGRAMACIÓN EN INTERNET: CLIENTES WEB

Internet ha revolucionado el mundo informático y, porque no, toda la sociedad en pocos años. Evidentemente, la programación de páginas web también ha evolucionado en los últimos años: se ha pasado de páginas sencillas, con pocas imágenes y que ofrecían contenidos estáticos a páginas complejas, con abundancia de imágenes y otros elementos multimedia y que ofrecen contenidos dinámicos adaptados a cada usuario individual. Las páginas web han pasado a convertirse en verdaderas aplicaciones web.

La programación de las aplicaciones web se encuadra dentro de las arquitecturas cliente/servidor. Por una parte, tenemos el cliente web (el navegador) que solicita servicios. Por otro lado, el servidor web que ofrece servicios y responde a las peticiones de los clientes.

Este libro contempla la programación de la parte cliente de las aplicaciones web. En el "mundo Internet" existen muchas tecnologías que se pueden emplear para programar los clientes web, como ActiveX, applet, Flash, VRML, etc., pero sólo dos son las más extendidas y se pueden considerar "el estándar": HTML y JavaScript. El libro se centra en estas dos tecnologías y presta una especial atención a la creación de formularios, la base para cualquier aplicación web.


I.S.B.N.: 84-8454-118-5
9788484541189

Internet ha revolucionado el mundo informático y, porque no, toda la sociedad en pocos años. Evidentemente, la programación de páginas web también ha evolucionado en los últimos años: se ha pasado de páginas sencillas, con pocas imágenes y que ofrecían contenidos estáticos a páginas complejas, con abundancia de imágenes y otros elementos multimedia y que ofrecen contenidos dinámicos adaptados a cada usuario individual. Las páginas web han pasado a convertirse en verdaderas aplicaciones web.

La programación de las aplicaciones web se encuadra dentro de las arquitecturas cliente/servidor. Por una parte, tenemos el cliente web (el navegador) que solicita servicios. Por otro lado, el servidor web que ofrece servicios y responde a las peticiones de los clientes.

Este libro contempla la programación de la parte cliente de las aplicaciones web. En el "mundo Internet" existen muchas tecnologías que se pueden emplear para programar los clientes web, como ActiveX, applet, Flash, VRML, etc., pero sólo dos son las más extendidas y se pueden considerar "el estándar": HTML y JavaScript. El libro se centra en estas dos tecnologías y presta una especial atención a la creación de formularios, la base para cualquier aplicación web.

Sergio Luján

Programación en Internet

Clientes Web

ePUB v1.0
gimli 20.10.11



Sergio Luján Mora

Ingeniero en Informática por la Universidad de Alicante. Ha impartido diversos cursos sobre Internet y las diversas tecnologías que se emplean en la programación de aplicaciones web (HTML, JavaScript, ASP y Java).

Durante un año ha trabajado en el Laboratorio Multimedia (mmlab) de la Universidad de Alicante, desarrollando aplicaciones para Internet e intranets.

Desde 1999 forma parte del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante. Las asignaturas que imparte en la actualidad son *Programación en Internet* y *Programación y Estructuras de Datos*.

NOTA DEL AUTOR

Título: Programación en Internet: Clientes WEB

Autor: Sergio Luján Mora

I.S.B.N.: 84-8454-118-5
Depósito Legal: A-1401-2001

Edita: Editorial Club Universitario
www.ecu.fm

Printed in Spain
Imprime: Imprenta Gamma - Telf.: 965 67 19 87
C/. Cottolengo, 25 - San Vicente (Alicante)
gamma@gamma.fm
www.gamma.fm

Prefacio

En pocos años, Internet ha invadido casi todos los aspectos de la vida. Podemos comunicarnos a través de Internet de distintas formas (correo electrónico, radio y televisión *online*, telefonía IP). Podemos comprar diversos productos en Internet (libros, discos, entradas de cine). Podemos conocer gente a través de Internet (*chat*, foros de discusión). Para que todo ello funcione, hacen falta profesionales especializados en "programación en Internet".

En este libro se trata una pequeña porción de la "programación en Internet": la programación de aplicaciones web. Las aplicaciones web se encuadran dentro de las arquitecturas cliente/servidor. En concreto, en este libro se estudia como programar la parte cliente, los clientes web. Las tecnologías que se contemplan son HTML y JavaScript. Existen otras tecnologías, como ActiveX o *applets*, pero no están estandarizadas como HTML y JavaScript.

Este libro se complementa con otro de próxima aparición que tratará la programación de aplicaciones web desde el lado del servidor. En él se mostrarán las tecnologías que se emplean para programar los servidores web: CGI, ASP, JSP, etc.

Evidentemente, existen programas que permiten crear páginas HTML sin tener ni idea de HTML. Pero igual que se puede conducir un coche sin tener ni idea de mecánica o pilotar un avión sin tener ni idea de aerodinámica o de motores a reacción, conviene conocer "lo que se tiene entre manos" para poder obtener el máximo partido. La programación de aplicaciones web no consiste sólo en crear páginas web muy bonitas, con muchas imágenes y colores; hay que validar la entrada del usuario, acceder a bases de datos, etc.

Este no es un libro sobre programación básica, donde se expliquen conceptos como variable, bucle de repetición, expresión lógica o recursividad. Es necesario poseer un nivel mínimo de programación para poder abordar los temas que tratan sobre JavaScript. Sin embargo, cualquiera que haya programado en algún lenguaje no tendrá problemas en asimilarlo.

El libro está estructurado en seis capítulos. Los dos primeros capítulos son introductorios: se repasan las arquitecturas cliente/servidor en general y se presenta un tipo concreto, las aplicaciones web. El tercer capítulo está dedicado al lenguaje HTML. En el cuarto capítulo se explican los lenguajes de *script* en general y en el siguiente capítulo se trata un lenguaje concreto: JavaScript. El libro finaliza con el modelo de objetos de documento, que permite acceder a los elementos de una página web desde un lenguaje de *script.* Por último, existen tres apéndices donde se resumen las etiquetas de HTML y se explica como trabajar con los colores en HTML y como depurar errores de JavaScript.

Aunque no figuran como autores (y no tienen "ni idea" de HTML o JavaScript),

sin la participación de mis padres este libro no existiría, ya que su apoyo a lo largo de bastantes años me ha permitido llegar a escribir este libro.

Me gustaría agradecer a Marisa la paciencia (¿infinita o ínfima?) que ha tenido todas las veces que he llegado tarde debido a este libro.

Por último, mando un saludo a mis antiguos compañeros del Laboratorio Multimedia (mmlab), donde me introduje en el mundo de Internet, y a mis actuales compañeros del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante.

Alicante, 8 de octubre de 2001 *Sergio Luján Mora*

Índice general

Índice de cuadros

Índice de figuras

Índice de acrónimos

ASP *Active Server Pages*

Tecnología de Microsoft que permite crear páginas web dinámicas en el servidor. Se puede decir que las páginas **ASP** son similares a los programas **CGI**. Las páginas **ASP** suelen estar programados en *VBScript*, aunque también se pueden programar en otros lenguajes.

ASCII *American Standard Code for Information Interchange*

Código binario utilizado para representar letras, números, símbolos, etc. A cada carácter se le asigna un número del 0 al 127 (7 bits). Por ejemplo, el código **ASCII** para la A mayúscula es 65. Existen códigos **ASCII** extendidos de 256 caracteres (8 bits), que permiten representar caracteres no ingleses como las vocales acentuadas o la eñe. Los caracteres de la parte superior (128 a 255) de estos códigos **ASCII** extendidos varían de uno a otro. Por ejemplo, uno de los más extendidos es ISO Latin-1 (oficialmente ISO-8859-1).

CGI *Common Gateway Interface*

Estándar que permite el intercambio de información entre un servidor y un programa externo al servidor. Un programa **CGI** es un programa preparado para recibir y enviar datos desde y hacia un servidor web según este estándar. Normalmente se programan en *C* o en *Perl*, aunque se puede usar cualquier lenguaje de propósito general.

DHTML *Dynamic HTML*

Conjunto de extensiones a **HTML** que permiten modificar el contenido de una página web en el cliente sin necesidad de establecer una nueva comunicación con el servidor. Se basa en el uso de **DOM** para acceder al contenido de la página.

DLL *Dynamic Link Library*

Fichero que almacena funciones ejecutables o datos que pueden ser usados por una aplicación en Microsoft Windows. Una **DLL** puede ser usada por varios programas a la vez y se carga en tiempo de ejecución (no en tiempo de compilación).

DOM *Document Object Model*

Especificación que define como se puede acceder a los objetos de un documento **HTML** (ventanas, imágenes, formularios) a través de un

lenguaje de *script*. Básicamente define un jerarquía de objetos. **DOM** se encuentra en proceso de estandarización por **W3C**. **DHTML** depende de **DOM** para cambiar dinámicamente el contenido de una página web. Desgraciadamente, los dos navegadores mavoritarios poseen distintos modelos de objetos.

ECMA *European Computer Manufacturers Association*

ECMA es una asociación internacional que establece estándares relacionados con sistemas de comunicación y de información.

GIF *Graphics Interchange Format*

Formato gráfico de mapas de bit desarrollado por COMPUSERVE. Incorpora compresión de datos, transparencias y animaciones. Existen dos versiones de este estándar gráfico: 87a y 89a.

HTML *HyperText Markup Language*

Lenguaje compuesto de una serie de etiquetas o marcas que permiten definir el contenido y la apariencia de las páginas web. Aunque se basa en **SGML**, no se puede considerar que sea un subconjunto. Existen cientos de etiquetas con diferentes atributos. **W3C** se encarga de su estandarización. El futuro sustituto de **HTML** es **XHTML.**

HTTP *HyperText Transfer Protocol*

Es el protocolo que emplea la **WWW**. Define cómo se tienen que crear y enviar los mensajes y qué acciones debe tomar el servidor y el navegador en respuesta a un comando. Es un protocolo *stateless* (sin estado), porque cada comando se ejecuta independientemente de los anteriores o de los posteriores. Actualmente, la mayoría de los servidores soportan **HTTP** 1.1 (**RFC** 2616 de junio de 1999). Una de las principales ventajas de esta versión es que soporta conexiones persistentes: una vez que el navegador se conecta al servidor, puede recibir múltiples ficheros a través de la misma conexión, lo que aumenta el rendimiento de la transmisión hasta en un 20 %.

ISAPI *Internet Server Application Program Interface*

Un API para el servidor Microsoft Internet Information Server. Permite programar aplicaciones web.

ISO *International Organization for Standards*

Organización fundada en 1946, cuyos miembros son las organizaciones nacionales de normalización (estandarización) correspondientes a los países

miembros. Entre sus miembros se incluyen a la ANSI (Estados Unidos), BSI (Gran Bretaña), AFNOR (Francia), DIN (Alemania) y UNE (España).

JPEG *Joint Photographic Experts Group*

Formato gráfico de mapas de bit. Incorpora compresión de datos con pérdidas y permite trabajar con 24 bits de color.

JSP *Java Server Pages*

Tecnología de SUN MICROSYSTEMS que permite crear páginas dinámicas en el servidor. Equivale a la tecnología **ASP** de Microsoft. Se programan en *Java*.

MIME *Multipurpose Internet Mail Extensions*

Se usa en el correo electrónico desde 1992 para enviar y recibir ficheros de distinto tipo. Se puede consultar el estándar en **RFC** 1341, **RFC** 1521 y **RFC** 1522.

PNG *Portable Network Graphics*

Formato gráfico de mapas de bit similar a **GIF**. **W3C** ha decidido sustituir **GIF** por **PNG** debido a que el primero emplea un algoritmo que está patentado, mientras que **PNG** es totalmente gratuito. Tanto Microsoft Internet Explorer como Netscape Communicator aceptan este formato.

RFC *Request for Comments*

Medio de publicar propuestas sobre Internet. Cada **RFC** recibe un número. Algunos se convierten en un estándar de Internet.

RGB *Red Green Blue*

Notación de los colores en la que cada color se representa como una combinación de los tres colores básicos (primarios) rojo (*red*), verde (*green*) y azul (*blue*). Se trata de un modelo aditivo (se parte del negro). Mediante la combinación adecuada de los tres colores básicos se consigue todo el espectro de colores. Además de **RGB** existen otras formas de representar los colores. Otra de las más corrientes es CMYK (*cyan*, *magenta*, *yellow*, *black*), que se trata de un modelo sustractivo.

SGML *Standard Generalized Markup Language*

Lenguaje que permite organizar y etiquetar los distintos elementos que componen un documento. Se emplea para manejar grandes documentos que sufren constantes revisiones y se imprimen en distintos formato. Desarrollado y estandarizado por **ISO** en 1986.

TCP/IP *Transmission Control Protocol/Internet Protocol*

Familia de protocolos que se emplean en las comunicaciones de Internet.

URL *Universal Resource Locator*

También conocido como *Uniform Resource Locator*. Sistema de direccio- namiento de máquinas y recursos en Internet. Es decir, se trata de una dirección que permite localizar cualquier máquina o documento que se encuentre accesible a través de Internet.

W3C *World Wide Web Consortium*

Consorcio internacional de compañías involucradas en el desarrollo de Internet y en especial de la **WWW**. Su propósito es desarrollar estándares y "poner orden" en Internet.

WWW *World Wide Web*

Sistema de servidores web conectados a Internet (no todos los ordenadores conectados a Internet forman parte de la **WWW**). Su protocolo de comunicación es **HTTP**, su lenguaje de creación de documentos **HTML** y sus sistema de direccionamiento de los recursos **URL**. Los navegadores web (*browsers*) permiten navegar por la web.

XHTML *Extensible HyperText Markup Language*

HTML escrito según las normas que marca **XML**. Por tanto, se trata de una aplicación concreta de **XML** y no tienen que confundirse entre sí.

XML *Extensibíe Markup Language*

Metalenguaje de etiquetado basado en **SGML**. Diseñado específicamente para la **WWW** por **W3C.** Permite que un usuario diseñe sus propias etiquetas, con sus atributos y las reglas de construcción de documentos (sintaxis).

Capítulo 1

Arquitecturas cliente/servidor

Las aplicaciones web son un tipo especial de aplicaciones cliente/servidor. Antes de aprender a programar aplicaciones web conviene conocer las características básicas de las arquitecturas cliente/servidor.

Índice General

1.1. Introducción

Cliente/servidor es una arquitectura de red‹ 1 › en la que cada ordenador o proceso en la red es **cliente** o **servidor**. Normalmente, los servidores son ordenadores potentes dedicados a gestionar unidades de disco (servidor de ficheros), impresoras (servidor de impresoras), tráfico de red (servidor de red), datos (servidor de bases de datos) o incluso aplicaciones (servidor de aplicaciones), mientras que los clientes son máquinas menos potentes y usan los recursos que ofrecen los servidores.

Esta arquitectura implica la existencia de una relación entre procesos que solicitan servicios (**clientes**) y procesos que responden a estos servicios (**servidores**). Estos dos tipos de procesos pueden ejecutarse en el mismo procesador o en distintos.

La arquitectura cliente/servidor implica la realización de aplicaciones distribuidas. La principal ventaja de esta arquitectura es que permite separar las funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada para su ejecución. Además, también presenta las siguientes ventajas:

- Las redes de ordenadores permiten que múltiples procesadores puedan ejecutar

partes distribuidas de una misma aplicación, logrando concurrencia de procesos.

- Existe la posibilidad de migrar aplicaciones de un procesador a otro con modificaciones mínimas en los programas.

- Se obtiene una escalabilidad de la aplicación. Permite la ampliación horizontal o vertical de las aplicaciones. La **escalabilidad horizontal** se refiere a la capacidad de añadir o suprimir estaciones de trabajo que hagan uso de la aplicación (clientes), sin que afecte sustancialmente al rendimiento general. La **escalabilidad vertical** permite la migración hacia servidores de mayor o menor capacidad y velocidad o de un tipo diferente.

- Posibilita el acceso a los datos independientemente de donde se encuentre el usuario.

1.2. Separación de funciones

La arquitectura cliente/servidor nos permite la separación de funciones en tres niveles, tal como se muestra en la Figura 1.1:

- **Lógica de presentación**. La presentación de los datos es una función independiente del resto.

- **Lógica de negocio (o aplicación)**. Los flujos de trabajo pueden cambiarse según las necesidades existentes de un procesador a otro.

- **Lógica de datos**. La gestión de los datos debe ser independiente para poder ser distribuida según las necesidades de la empresa en cada momento.

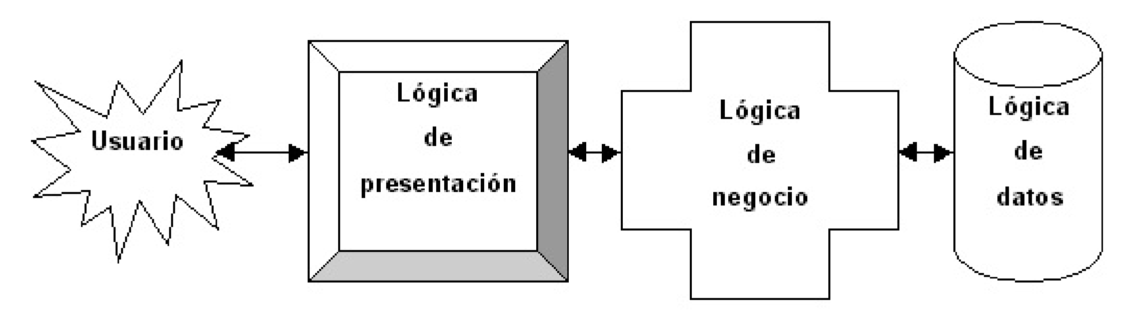


Figura 1.1: Separación de funciones

Si un sistema distribuido se diseña correctamente, los tres niveles anteriores pueden distribuirse y redistribuirse independientemente sin afectar al funcionamiento de la aplicación.

1.3. Modelos de distribución en aplicaciones cliente/servidor

Según como se distribuyan las tres funciones básicas de una aplicación (presentación, negocio y datos) entre el cliente y el servidor, podemos contemplar tres modelos: presentación distribuida, aplicación distribuida y datos distribuidos.

1.3.1. Presentación distribuida

El cliente sólo mantiene la presentación, el resto de la aplicación se ejecuta remotamente (Figura 1.2). La presentación distribuida, en su forma más simple, es una interfaz gráfica de usuario a la que se le pueden acoplar controles de validación de datos, para evitar la validación de los mismos en el servidor.

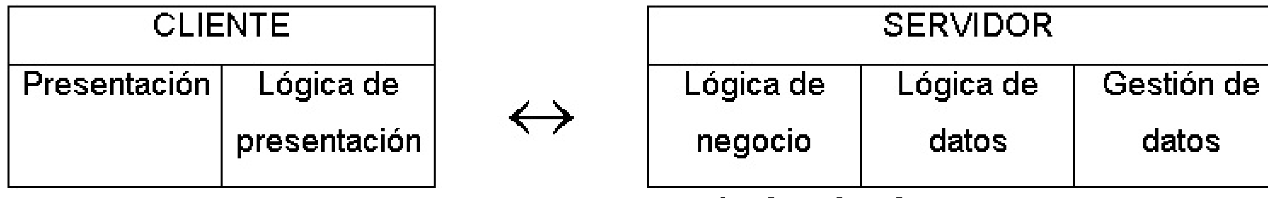


Figura 1.2: Presentación distribuida

1.3.2. Aplicación distribuida

Es el modelo que proporciona máxima flexibilidad, puesto que permite tanto a servidor como a cliente mantener la lógica de negocio realizando cada uno las funciones que le sean más propias, bien por organización, o bien por mejora en el rendimiento del sistema (Figura 1.3).

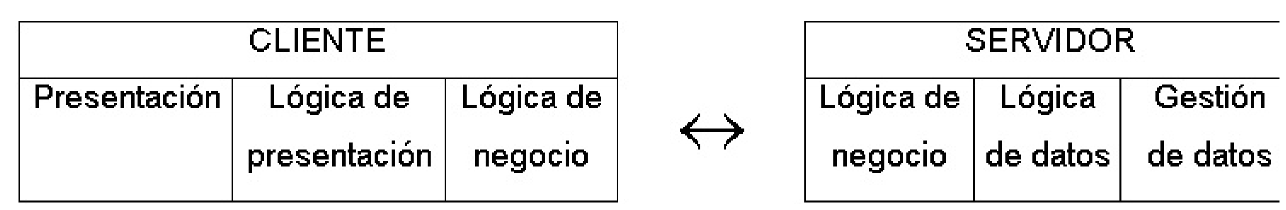


Figura 1.3: Aplicación distribuida

1.3.3. Datos distribuidos

Los datos son los que se distribuyen, por lo que la lógica de datos es lo que queda separada del resto de la aplicación (Figura 1.4). Se puede dar de dos formas: ficheros distribuidos o bases de datos distribuidas.

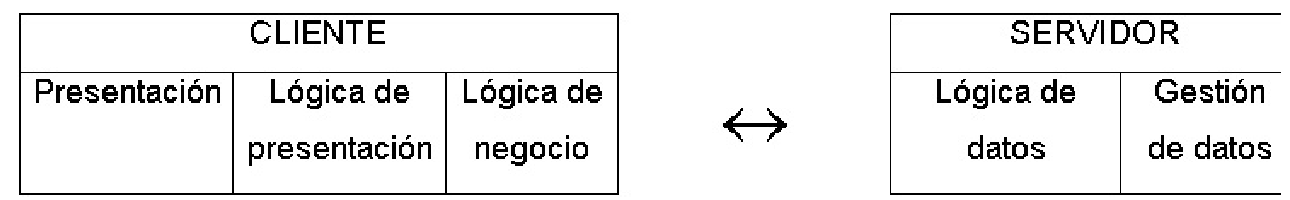


Figura 1.4: Datos distribuidos

1.4. Arquitecturas de dos y tres niveles

La diferencia entre las aplicaciones de dos y tres niveles estriba en la forma de distribución de la aplicación entre el cliente y el servidor.

Una arquitectura de dos niveles está basada en un sistema gestor de bases de datos donde el cliente mantiene la lógica de la presentación, negocio, y de acceso a los datos, y el servidor únicamente gestiona los datos. Suelen ser aplicaciones cerradas que supeditan la lógica de los procesos cliente al gestor de base de datos que se está usando.

En las arquitecturas de tres niveles, la lógica de presentación, la lógica de negocio y la lógica de datos están separadas, de tal forma que mientras la lógica de presentación se ejecutará normalmente en la estación cliente, la lógica de negocio y la de datos pueden estar repartidas entre distintos procesadores.

El objetivo de aumentar el número de niveles en una aplicación distribuida es lograr una mayor independencia entre un nivel y otro, lo que facilita la portabilidad en entornos heterogéneos.

1.5. Descripción de un sistema cliente/servidor

Un sistema cliente/servidor suele presentar las siguientes características:

1. Una combinación de la parte cliente (también llamada front-end) que interactúa con el usuario (hace de interfaz entre el usuario y el resto de la aplicación) y la parte servidor (o back-end) que interactúa con los recursos compartidos (bases de datos, impresoras, módems).

2. La parte cliente y servidor tienen diferentes necesidades de recursos a la hora de ejecutarse: velocidad de procesador, memoria, velocidad y capacidad de los discos duros, dispositivos de entrada/salida, etc.

3. El entorno suele ser heterogéneo y multivendedor. El hardware y sistema operativo del cliente y el servidor suelen diferir. El cliente y el servidor se suelen comunicar a través de unas API‹ 2 › y RPC‹ 3 › conocidas (por ejemplo, ODBC‹ 4 › para acceder a bases de datos).

4. Normalmente la parte cliente se implementa haciendo uso de una inter- faz gráfica de usuario, que permite la introducción de datos a través de teclado, ratón, lápiz óptico, etc.

Capítulo 2

Qué es una aplicación web

En las aplicaciones web suelen distinguirse tres niveles (como en las arquitecturas cliente/servidor de tres niveles): el nivel superior que interacciona con el usuario (el cliente web, normalmente un navegador), el nivel inferior que proporciona los datos (la base de datos) y el nivel intermedio que procesa los datos (el servidor web). En este capítulo se describen el cliente y el servidor web y se comentan los entornos web en los que se ejecutan las aplicaciones web.

Índice General

2.1. Introducción

Una aplicación web (*web-based application*) es un tipo especial de aplicación cliente/servidor, donde tanto el **cliente** (el navegador, explorador o visualizador‹ 5 ›) como el **servidor** (el servidor web) y el **protocolo** mediante el que se comunican (*HyperText Transfer Protocol* (**HTTP**)) están estandarizados y no han de ser creados por el programador de aplicaciones (Figura 2.1).

El protocolo **HTTP** forma parte de la familia de protocolos de comunicaciones *Transmission Control Protocol/Internet Protocol* (**TCP/IP**), que son los empleados en Internet. Estos protocolos permiten la conexión de sistemas heterogéneos, lo que facilita el intercambio de información entre distintos ordenadores.

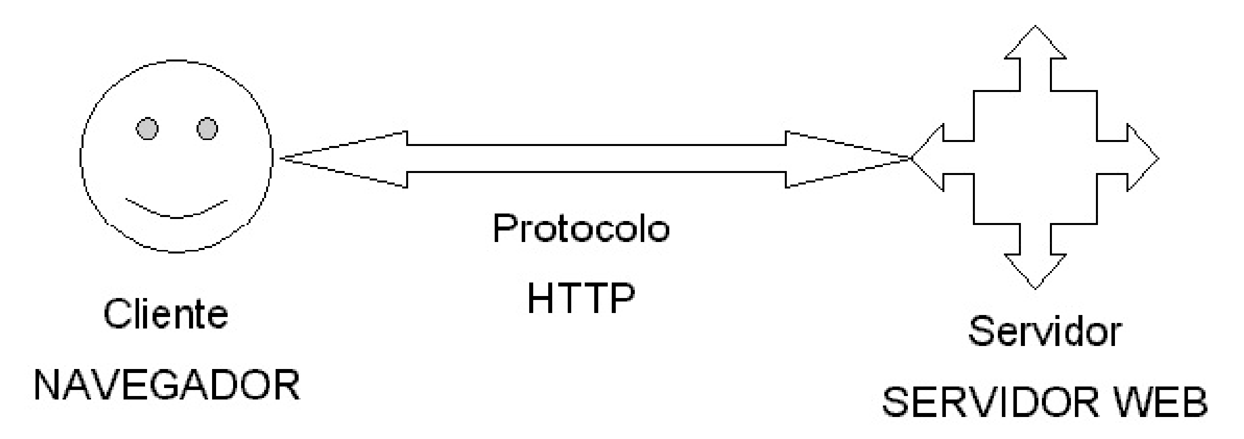


Figura 2.1: Esquema básico de una aplicación web

2.1.1. El cliente

El cliente web es un programa con el que interacciona el usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante **HTTP**‹ 6 ›.

La parte cliente de las aplicaciones web suele estar formada por el código *HyperText Markup Language* (**HTML**) que forma la página web más algo de código ejecutable realizado en lenguaje de *script* del navegador (*JavaScript* o *VBScript*) o mediante pequeños programas (*applets*) realizados en *Java*. También se suelen emplear *plug-ins*‹ 7 › que permiten visualizar otros contenidos multimedia (como Flash‹ 8 ›), aunque no se encuentran tan extendidos como las tecnologías anteriores y plantean problemas de incompatibilidad entre distintas plataformas. Por tanto, la misión del cliente web es interpretar las páginas **HTML** y los diferentes recursos que contienen (imágenes, sonidos, etc.).

2.1.2. El servidor

El servidor web es un programa que está esperando permanentemente las solicitudes de conexión mediante el protocolo **HTTP** por parte de los clientes web. En los sistemas Unix suele ser un "demonio" y en los sistemas Microsoft Windows un servicio.

La parte servidor de las aplicaciones web está formada por páginas estáticas que siempre muestran el mismo contenido y por programas o *scripts* que son ejecutados por el servidor web cuando el navegador del cliente solicita algunas páginas. La salida de este *script* suele ser una página **HTML** estándar que se envía al navegador

del cliente. Tradicionalmente este programa o *script* que es ejecutado por el servidor web se basa en la tecnología *Common Gateway Interface* (**CGI**).

La programación del servidor mediante **CGI** es compleja y laboriosa. El protocolo **HTTP** no almacena el estado entre una conexión y la siguiente (es un protocolo sin estado), por lo que es el programador el que se tiene que encargar de conservarlo. Esto conduce a que el programador tenga que dedicar parte de su tiempo a programar tareas ajenas al n de la aplicación, lo que suele ser origen de diversos problemas.

Sin embargo, con la entrada en 1995 de MICROSOFT en el mundo Internet y la salida al mercado de su servidor web (Internet Information Server) se abrió un nuevo campo para las aplicaciones web: *Internet Server Application Program Interface* (**ISAPI**). Se trata de un conjunto de funciones que el servidor web pone a disposición de los programadores de aplicaciones web. Con **ISAPI**, los programadores pueden crear *Dynamic Link Library* (**DLL**) con funciones que son invocadas para determinados archivos (se ejecutan cuando el cliente solicita un archivo con una determinada extensión).

Todo el sistema *Active Server Pages* (**ASP**), no es más que una **DLL** del tipo **ISAPI** que es invocada automáticamente para los archivos cuya extensión sea .asp. La DLL ASP preprocesa el archivo .asp interpretando su código como un *script* a ejecutar en el servidor. Sin embargo, ella no interpreta directamente el código, sino que en función del lenguaje en el que está escrito, invoca a otra **DLL** que se encarga de ejecutar el *script*. Después recoge la salida y se la envía al servidor web, el cual a su vez la reenvía al cliente.

Las ventajas que presenta **ASP** frente a **CGI** son:

- Las páginas basadas en **CGI** resultan difíciles de mantener, ya que las instrucciones **HTML** se encuentran insertadas en el propio código del programa CGI, mezclándose sus funcionalidades.

- La ejecución de un programa **CGI** es muy ineciente, debido al proceso de carga del código en memoria que se realiza cada vez que un usuario requiere su ejecución. La existencia de múltiples clientes simultáneos supone múltiples copias del programa en memoria del servidor.

- La unión de **ISAPI** con el servidor web es más "fuerte" (están más integrados), su ejecución es más rápida, con lo que se logra que las aplicaciones basadas en **ISAPI** tengan un mayor rendimiento que las basadas en **CGI**.

Además de **ASP**, existen otras tecnologías destinadas a programar la parte servidor de las aplicaciones web: *ColdFusion*, *Java Server Pages* (**JSP**), *servlets*, ***PHP***, etc. Todas ellas son muy similares, se basan en los mismos principios y ofrecen

resultados equivalentes.

2.2. Transferencia de páginas web

El proceso completo, desde que el usuario solicita una página, hasta que el cliente web (navegador) se la muestra con el formato apropiado, es el siguiente:

1. El usuario especifica en el cliente web la dirección de la página que desea consultar: el usuario escribe en el navegador la dirección (*Universal Resource Locator* (**URL**)) de la página que desea visitar.
2. El cliente establece una conexión con el servidor web.
3. El cliente solicita la página o el objeto deseado.
4. El servidor envía dicha página u objeto (o, si no existe, devuelve un código de error).
5. Si se trata de una página **HTML**, el cliente inicia sus labores de interpretación de los códigos **HTML**. Si el cliente web encuentra instrucciones que hacen referencia a otros objetos que se tienen que mostrar con la página (imágenes, sonidos, animaciones multimedia, etc.), establece automáticamente comunicación con el servidor web para solicitar dichos objetos.
6. Se cierra la conexión entre el cliente y el servidor.
7. Se muestra la página al usuario.

Obsérvese que siempre se libera la conexión, por lo que ésta sólo tiene laduración correspondiente a la transmisión de la página solicitada. Esto se hace así para no desperdiciar innecesariamente el ancho de banda de la red mientras el usuario lee la página recibida.

Cuando el usuario activa un enlace de la página, se establece una nueva conexión para recibir otra página o elemento multimedia. Por ello, el usuario tiene la sensación de que está disfrutando de una conexión permanente cuando realmente no es así.

Un detalle importante es que para casa objeto que se transere por la red se realiza una conexión independiente. Por ejemplo, si el cliente web solicita una página que contiene dos imágenes integradas, se realizan tres conexiones: una para el documento **HTML** y dos para los archivos de las imágenes.

2.3. Entornos web

Las aplicaciones web se emplean en tres entornos informáticos muy similares que suelen confundirse entre sí: *Internet*, *intranet* y *extranet*.

2.3.1. Internet

Internet es una red global que conecta millones de ordenadores por todo el mundo. Su nacimiento se sitúa en 1969, en un proyecto de investigación del Departamento de Defensa de Estados Unidos. En 1998, la Internet tenía más de 100 millones de usuarios en todo el mundo, en diciembre de 2000 unos 400 millones y el número sigue creciendo rápidamente. Más de 100 países están conectados a este nuevo medio para intercambiar todo tipo de información.

Al contrario que otros servicios *online*, que se controlan de forma centralizada, la Internet posee un diseño descentralizado. Cada ordenador (*host*) en la Internet es independiente. Sus operadores pueden elegir que servicio de Internet usar y que servicios locales quieren proporcionar al resto de la Internet. Asombrosamente, este diseño anárquico funciona satisfactoriamente.

Existe una gran variedad de formas de acceder a la Internet. El método más común es obtener acceso a través de Proveedores de servicios de Internet (*Internet Service Provider*, **ISP**).

Cuando se emplea la palabra internet en minúsculas, nos referimos a un conjunto de dos o más redes de ordenadores interconectadas entre sí.

2.3.2. Intranet

Una intranet es una red de ordenadores basada en los protocolos que gobiernan Internet (**TCP/IP**) que pertenece a una organización y que es accesible únicamente por los miembros de la organización, empleados u otras personas con autorización.

Una intranet puede estar o no conectada a Internet. Un sitio web en una intranet es y actúa como cualquier otro sitio web, pero los cortafuegos (*firewall*) lo protegen de accesos no autorizados.

Al igual que Internet, las intranets se usan para distribuir y compartir información. Las intrantes hoy en día componen el segmento con el mayor

crecimiento dentro de Internet, porque son menos caras de montar y de administrar que las redes privadas que se basan en protocolos propietarios.

2.3.3. Extranet

Una extranet es una intranet a la que pueden acceder parcialmente personas autorizadas ajenas a la organización o empresa propietaria de la intranet.

Mientras que una intranet reside detrás de un cortafuego y sólo es accesible por las personas que forman parte de la organización propietaria de la intranet, una extranet proporciona diferentes niveles de acceso a personas que se encuentran en el exterior de la organización. Esos usuarios pueden acceder a la extranet sólo si poseen un nombre de usuario y una contraseña con los que identificarse. La identidad del usuario determina que partes de la extranet puede visualizar.

Las extranets se están convirtiendo en un medio muy usado por empresas que colaboran para compartir información entre ellas. Se emplean como medio de comunicación de una empresa con sus clientes, proveedores o socios. Las extranets son la base del comercio entre empresas (*business to business*, **B2B**).

2.4. Ventajas y desventajas

El desarrollo explosivo de Internet y en especial de la WWW se debe a la aceptación por todo el mundo de los estándares y tecnologías que emplea: medio de transporte común (**TCP/IP**), servidor (**HTTP**) y lenguaje de creación de páginas (**HTML**) estandarizados.

Muchas empresas han descubierto que las anteriores tecnologías se puedenemplear en las aplicaciones cliente/servidor que emplean. De esta forma nace el concepto de intranet: usar las tecnologías de Internet para implementar las tradicionales aplicaciones cliente/servidor dentro de una empresa. Además, una vez que se tiene una aplicación que funciona en una intranet, aparece la posibilidad de permitir su uso a través de Internet, lo que facilita el teletrabajo o la movilidad de los empleados de una empresa‹ 9 ›.

Una ventaja clave del uso de aplicaciones web es que el problema de gestionar el código en el cliente se reduce drásticamente. Suponiendo que existe un navegador o explorador estándar en cada cliente, todos los cambios, tanto de interfaz como de funcionalidad, que se deseen realizar a la aplicación se realizan cambiando el código que resida en el servidor web. Compárese esto con el coste de tener que actualizar

uno por uno el código en cada uno de los clientes (imaginemos que tenemos 2.000 ordenadores clientes). No sólo se ahorra tiempo porque reducimos la actualización a una sólo máquina, sino que no hay que desplazarse de un puesto de trabajo a otro (la empresa puede tener una distribución geográfica amplia).

Una segunda ventaja, relacionada con la anterior, es que se evita la gestión de versiones. Se evitan problemas de inconsistencia en las actualizaciones, ya que no existen clientes con distintas versiones de la aplicación.

Una tercera ventaja es que si la empresa ya está usando Internet, no se necesita comprar ni instalar herramientas adicionales para los clientes.

Otra ventaja, es que de cara al usuario, los servidores externos (Internet) e internos (intranet) aparecen integrados, lo que facilita el aprendizaje y uso.

Una última ventaja, pero no menos importante, es la independencia de plataforma. Para que una aplicación web se pueda ejecutar en distintas plataformas (*hardware* y sistema operativo), sólo se necesita disponer de un navegador para cada una de las plataformas, y no es necesario adaptar el código de la aplicación a cada una de ellas. Además, las aplicaciones web ofrecen una interfaz gráfica de usuario independiente de la plataforma (ya que la plataforma de ejecución es el propio navegador).

Una desventaja, que sin embargo está desapareciendo rápidamente, es que la programación en la web no es tan versátil o potente como la tradicional. Al principio, las aplicaciones web eran básicamente de "solo lectura": permitían una interacción con el usuario prácticamente nula. Sin embargo, con la aparición de nuevas herramientas como *Java*, *JavaScript* y **ASP**, esta limitación tiende a desaparecer.

Capítulo 3

HTML

HTML es un lenguaje de marcas (etiquetas) que se emplea para dar formato a los documentos que se quieren publicar en la WWW. Los navegadores pueden interpretar las etiquetas y muestran los documentos con el formato deseado. En este capítulo se presentan los conceptos básicos y avanzados (tablas, formularios y marcos) de HTML. El capítulo finaliza con una guía de estilo: pequeños consejos que pueden ayudar a lograr mejores páginas web.

Índice General

3.1. Introducción

Las páginas web o páginas **HTML** son unos ficheros escritos en el lenguaje **HTML**. El desarrollo de estas páginas abarca un amplio grupo de tecnologías, desde las páginas más sencillas que sólo usan el lenguaje **HTML** hasta las más complejas que usan *Dynamic HTML* (**DHTML**), *JavaScript*, *applets* realizados en *Java* o componentes *ActiveX*.

El lenguaje **HTML** se basa en *Standard Generalized Markup Language* (**SGML**), un sistema mucho más completo y complicado de procesamiento de documentos que indica como organizar y marcar (etiquetar) un documento. **HTML**

define e interpreta las etiquetas de acuerdo a **SGML**.

Las páginas **HTML** se pueden diseñar usando texto con distintos tipos de letras o colores, imágenes, listas de elementos, tablas, etc. Su modo de empleo es muy sencillo: se basa en el uso de etiquetas que indican que elementos contiene cada página, el formato que hay que aplicar a cada uno de ellos y como se tienen que distribuir por la página.

3.2. Evolución de HTML

El nacimiento de **HTML** va ligado al de la *World Wide Web* (**WWW**). Los orígenes de ambos se sitúan en 1991, en los trabajos que llevaba a cabo Tim Berners-Lee y sus compañeros en el CERN‹ 10 › en Suiza. Uno de los primeros artículos en los que muestran sus ideas es "World-Wide Web: The Information Universe"‹ 11 ›. En este artículo detallan un sistema que permita realizar el sueño de "interconectar todo el conocimiento de la humanidad y facilitar su acceso a todo el mundo gracias al empleo de los ordenadores". Para lograrlo, hacen uso de tecnologías como el hipertexto o la hipermedia, tecnologías que ya existían desde hacía varios años (en contra de lo que la gente cree, estas tecnologías no fueron inventadas por ellos).

Entre las distintas partes que componen el sistema que proponen, se incluye "una sintaxis en el estilo de SGML" para proporcionar formato a los documentos. A partir de ahí nace **HTML** como un lenguaje para el intercambio de documentos científicos y técnicos. **HTML** evita la complejidad de **SGML** al definir un pequeño conjunto de etiquetas que simplifican la estructura de los documentos y las reglas no son tan estrictas como en **SGML**.

En octubre de 1994, Tim Berners-Lee, funda el *World Wide Web Consortium* (**W3C**) en el *Massachusetts Institute of Technology, Laboratory for Computer Science [MIT/LCS]* en colaboración con el CERN, y con el apoyo de DARPA‹ 12 › y de la Comisión Europea. En abril de 1995, el INRIA‹ 13 › se convierte en el primer *host* europeo de **W3C**.

El objetivo principal del **W3C** es encabezar el desarrollo de la **WWW**, mediante la elaboración de protocolos que aseguren su estandarización. Hoy en día, **W3C** lidera el desarrollo de distintas tecnologías, como **HTML**, **HTTP**, *Extensible Markup Language* (**XML**), *Portable Network Graphics* (**PNG**), etc. En el Cuadro 3.1 mostramos las distintas versiones de **HTML** que se han estandarizado‹ 14 › desde 1995.

Fecha	Versión

Noviembre 1995	HTML 2.0
Enero 1997	HTML 4.0
Diciembre 1997	HTML 3.2
Diciembre 1999	HTML 4.01
Enero 2000	XHTML 1.0

Cuadro 3.1: Versiones de HTML

En enero de 2000 aparece *Extensible HyperText Markup Language* (**XHTML**) 1.0, el futuro sustituto de **HTML**. Como dice el propio estándar, se trata de "una reformulación de HTML en XML 1.0". **XHTML** es el lenguaje **HTML** escrito según las normas que impone **XML**. Por tanto, es una aplicación concreta de **XML** y no deben confundirse entre sí. Las principales diferencias entre **HTML** y **XHTML** 1.0 son:

- Las etiquetas y atributos tienen que escribirse en minúsculas.
- Los valores de los atributos tienen que ir entre comillas.
- No se admiten atributos sin valor.
- Todas las etiquetas tienen que aparecer por parejas (inicio y fin) o como etiquetas vacías.

3.3. Clasificación de las páginas

Según como se generan las páginas web en el servidor, se clasifican en:

- **Estáticas**. Poseen un contenido fijo, todos los usuarios que las consultan reciben la misma información. El usuario recibe en su navegador la página del servidor sin un procesamiento previo‹ 15 ›.
- **Dinámicas o activas en el servidor**: poseen un contenido variable, distintos usuarios al consultar la misma página pueden recibir distintos contenidos. El usuario recibe en su navegador la página después de haber sido procesada en el servidor. Para lograrlo se emplean lenguajes de programación. Ejemplo: páginas generadas por un **CGI**, páginas **ASP**, etc.

Por otro lado, según como se visualizan las páginas en el cliente, se clasifican en:

- **Estáticas**. Cuando no poseen ningún tipo de código de *script*, *applets* o *plug-ins*. Ejemplo: sólo código **HTML**.
- **Dinámicas o activas en el cliente**. Cuando se interpreta o ejecuta código en el equipo del usuario. Para lograrlo se emplean lenguajes de programación y objetos integrados. Ejemplo: páginas con *JavaScript*, **DHTML**, *applets*, etc.

Las características anteriores se pueden combinar como se quieran: una página puede ser estática en el servidor y en el cliente, estática en el servidor pero dinámica en el cliente, dinámica en el servidor y estática en el cliente y, por último, dinámica en el servidor y dinámica en el cliente.

Si nos planteamos crear páginas dinámicas en el servidor o en el cliente, surgen una serie de ventajas y desventajas. Las ventajas principales de las páginas activas en el cliente son:

- Se descarga de trabajo al servidor, ya que la ejecución del código se realiza de forma distribuida en cada cliente.
- Se reduce el ancho de banda necesario, ya que se evitan continuos traspasos de información del servidor al cliente y viceversa.
- Ofrecen respuestas inmediatas al usuario.

Sin embargo, existen algunas razones que nos llevan a crear páginas activas en el servidor:

- Cuando la información sobre la que se realiza el procesamiento es muy amplia, su envío al cliente supone un ancho de banda grande.
- Puede existir información restringida sobre la que se realice el procesamiento y que no interese que pueda llegar íntegra al cliente.
- Las páginas activas en el cliente se basan en tecnologías dependientes del navegador y del sistema operativo del usuario. Presuponer que el usuario dispone de los requisitos necesarios para que funcionen las páginas es un error.
- Las páginas activas en el cliente pueden ser poco seguras.

Entonces, ¿cuál de las cuatro combinaciones posibles elegir? La mejor opción es la última: páginas dinámicas tanto en el servidor como en el cliente, ya que podremos aprovechar las ventajas de las dos opciones y eliminar sus desventajas. Simplemente, habrá que distribuir la lógica de nuestra aplicación entre el servidor y el cliente, de forma que se obtenga la solución más óptima a nuestro problema.

En este capítulo nos vamos a centrar en el lenguaje **HTML**, que nos permite

realizar páginas estáticas en el cliente. En el Capítulo 5 veremos *JavaScript*, que nos permitirá realizar páginas dinámicas en el cliente. Ambas tecnologías no influyen en la parte del servidor. El desarrollo de páginas activas en el servidor queda fuera de los objetivos de este libro.

3.4. Qué necesito para usar HTML

No es necesario un servidor web, un proveedor web o tener una conexión a Internet para empezar a escribir documentos **HTML**. Los documentos **HTML** tienen un formato de texto plano (*American Standard Code for Information Interchange* (**ASCII**)), por lo que todo lo que se necesita es un editor (como el Bloc de notas de Microsoft Windows) para crear las páginas y un navegador para verlas. Podemos crear, vincular y probar documentos **HTML** completos en nuestro ordenador, aunque no esté conectado a ninguna red.

Para facilitar la creación de páginas **HTML**, han aparecido gran cantidad de programas. Básicamente, se pueden dividir en dos grupos: los editores de **HTML** y los programas de diseño **HTML**.

La mayoría de editores que ayudan a escribir **HTML** son simples editores de texto con algunos botones que pegan las etiquetas más comunes. Otros, suelen incluir la característica *syntax highlight*: significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean etiquetas, atributos, comentarios, etc.

Por otro lado, los programas de diseño muestran la página **HTML** de forma gráfica y en tiempo real: es posible desplazar los distintos elementos que la componen (tablas, imágenes, texto), modificar sus propiedades (tamaño, color, tipo de letra) y crear efectos avanzados. Un inconveniente de estos programas es que generan mucho código **HTML**: si en el futuro se desea modificar la página directamente a través del código **HTML**, es prácticamente imposible. Entre los mejores programas de esta clase destacan Adobe Golive, Claris Home Page, Macromedia DreamWeaver y Microsoft FrontPage.

Conviene aclarar desde un principio que lo único que da formato a una página web es una etiqueta **HTML**. Si editamos con cuidado un archivo de texto con objeto de tener párrafos y columnas de cifras formateadas, pero no se incluye ninguna etiqueta, al visualizarlo en **HTML** todo el texto fluirá en un solo párrafo y se perderá todo el formato que le hayamos aplicado.

La extensión de un archivo **HTML** suele ser .html o .htm‹ 16 ›. Se deben emplear nombres cortos y sencillos. Hay que evitar el uso de espacios o de caracteres especiales en el nombre del archivo y también controlar el uso de mayúsculas y

minúsculas puesto que en Internet existen multitud de sistemas operativos, que no pueden aceptar los mismos nombres de archivo que acepta el nuestro. Por ejemplo, hay sistemas operativos en los que las mayúsculas y minúsculas se distinguen (Unix) y otros donde no (Microsoft Windows‹ 17 ›).

3.5. Conceptos básicos de HTML

El lenguaje **HTML** consta de una serie de etiquetas o marcas (*tags*). La mayoría de las etiquetas aparecen por parejas (códigos pareados), siendo una de comienzo (apertura) y otra de fin‹ 18 › (cierre): delimitan la parte del documento **HTML** que se ve afectada por su acción. Pero también hay etiquetas que aparecen de forma individual, como <IMG> para insertar una imagen.

Todas las etiquetas comienzan con el símbolo < (menor que) y terminan con el símbolo > (mayor que). Entre estos dos símbolos aparece el nombre de la etiqueta. Por ejemplo, <HR> es una etiqueta válida, pero <HR o <HR< no lo son.

Las etiquetas de fin tienen el mismo nombre que las de inicio, pero van precedidas del símbolo / (barra inclinada). Por ejemplo, la etiqueta de cierre correspondiente a <HTML> es </HTML>.

Una etiqueta puede poseer varios atributos a los que hay que asignar valor. Estos atributos suelen ser opcionales y algunos necesitan un tipo de valor concreto. Los atributos se escriben dentro de la etiqueta y separados por espacios en blanco. Para asignar un valor a un atributo se emplea el signo igual (=). Por ejemplo, la etiqueta <IMG>, que no tiene una etiqueta de cierre, tiene varios atributos, entre ellos WIDTH y HEIGHT que esperan un valor numérico y el atributo ALT que espera cualquier cadena de caracteres.

Cuando un usuario solicita una página **HTML** a un servidor web, este envía la página tal cual. En el momento en que el explorador recibe la página, interpreta las etiquetas que la página contiene, mostrando al usuario el resultado final (donde no aparecen ya las etiquetas, sino el resultado de su evaluación).

3.5.1. Estructura de una página

La estructura básica de una página se divide en cabecera (<HEAD> ... </HEAD>) y cuerpo (<BODY> ... </BODY>). El esqueleto básico de una página es:

Ejemplo 3.1

```
1 <!DOCTYPE HTML PUBLIC ".//W3C//DTD HTML 4.0//EN">
2 <HTML>
3 <HEAD>
4 Cabecera de la página
5 </HEAD>
6 <BODY>
7 Cuerpo de la página
8 </BODY>
9 </HTML>
```

El sentido de cada una de las líneas es:

- Línea 1: Permite indicar la versión **HTML** que se va a utilizar para escribir la página. Normalmente no se incluye.
- Línea 2: Junto a la línea 9, indican el comienzo y fin de la página. La etiqueta <HTML> es obligatoria.
- Línea 3: Junto a la línea 5, indican el comienzo y fin de la cabecera.
- Línea 4: Lo que escribamos en la cabecera no se verá en la página. Se suele usar para indicar el título de la página con <TITLE> ... </TITLE>, incluir código que se ejecuta en el cliente con <SCRIPT> ... </SCRIPT>, definir un estilo con <STYLE> ... </STYLE> o incluir información sobre el contenido de la página con la etiqueta <META>.
- Línea 6: Junto a la línea 8, indican el comienzo y fin del cuerpo.
- Línea 7: Lo que escribamos en el cuerpo se verá en la página. La información contenida en esta sección se puede visualizar con apariencias muy diversas: sólo hay que aplicar distintos formatos a las secciones que la componen.

Cuando se escriban las etiquetas de fin hay que llevar mucho cuidado en el orden. Por ejemplo, en el esqueleto anterior, la etiqueta <HTML> aparece antes que <BODY>; la última etiqueta en aparecer es la primera que se tiene que cerrar. Por tanto, el orden correcto es primero </BODY> y luego </HTML>.

El siguiente código crea una página **HTML** sencilla, que podemos ver en la Figura 3.1. El título indicado sólo aparece en la barra de título de la ventana activa del navegador. El título se indica usando los código pareados <TITLE> ... </TITLE>, y se especifica usando un texto plano, sin códigos **HTML** de formato (no se puede poner en negrita o cursiva, por ejemplo).

Ejemplo 3.2

```
<HTML>
<HEAD>
```

```
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
Esto es el cuerpo de una página HTML
<BR>
Esto tiene que aparecer en una línea nueva
</BODY>
</HTML>
```

Es muy aconsejable poner título a todos los documentos y, además, se deberá procurar que éste sea lo más descriptivo posible, puesto que si algún usuario decide incluir nuestra página en su lista de enlaces (*bookmarks*), será el título lo que quede almacenado en dicha lista (junto con la **URL**). En el caso de que no se haya especificado un título, lo que se almacena en la lista de enlaces es la dirección de la página (**URL**).



Figura 3.1: Primera página HTML

En cualquier parte de una página **HTML**, si queremos incluir un comentario deberemos emplear la etiqueta <!-- Comentario -->. Los comentarios no se procesan y no producen una salida visible en la página.

Los saltos de línea que incluyamos en una página son irrelevantes (el navegador no los tiene en cuenta). Para producir un salto de línea se emplea la etiqueta
. Los espacios en blanco también son irrelevantes: si separamos dos palabras por varios espacios en blanco, sólo se tendrá en cuenta uno de ellos. Si queremos incluir varios espacios en blanco, debemos de emplear el código de escape . Por ejemplo, para incluir tres espacios en blanco se tiene que escribir   .

3.5.2. Caracteres especiales y secuencias de escape

Algunos sistemas informáticos trabajan con 7 bits en vez de 8. En esos casos, si

se desea trabajar con caracteres **ASCII** de la parte superior de la tabla (128-255), es necesario codificarlos de algún modo. En los documentos **HTML** se pueden codificar de dos formas, mediante referencias decimales o referencias a entidades.

Las referencias decimales (también llamadas secuencias de escape) usan el formato &#nnn; donde nnn es el código **ASCII** del carácter. Por ejemplo, Á representa el carácter Á.

Las referencias a entidades usan el formato &aaaa; donde aaaa es una cadena de texto que representa el carácter. Por ejemplo, Á representa el carácter Á.

Además, existen algunos caracteres de la parte inferior de la tabla de caracteres **ASCII** que poseen un significado especial en **HTML**, por lo que es necesario codificarlos. En el Cuadro 3.2 figuran los caracteres especiales con su secuencia de escape.

Carácter	Decimal	Entidad
"	"	"
&	&	&
<	<	<
>	>	>

Cuadro 3.2: Caracteres con un significado especial en HTML

También existen otros caracteres que son difíciles de conseguir desde el teclado, ya que no están representados. En el Cuadro 3.3 figuran algunos de los más significativos.

Carácter	Decimal	Entidad
§	§	§
©	©	©
®	®	®
¶	¶	¶

Cuadro 3.3: Caracteres especiales

3.6. Metadatos

La etiqueta <META> permite indicar metadatos‹ 19 › de una forma muy simple. Permite incorporar información sobre el contenido de una página. Esta etiqueta sólo puede aparecer en la sección <HEAD>. Esta etiqueta la emplean los motores de búsqueda, los navegadores y las herramientas de diseño de páginas web.

La etiqueta <META> presenta dos variantes. La sintaxis de estas dos variantes es:

- <META HTTP-EQUIV="propiedad" CONTENT="contenido">. El atributo HTTP-EQUIV se emplea cuando la página web se recupera mediante **HTTP**. Los servidores web pueden usar el nombre de la propiedad para crear una cabecera según el estándar *Request for Comments* (**RFC**) 822‹ 20 › en la cabecera de la respuesta HTTP (algunas propiedades de la cabecera no se pueden fijar de esta forma). Por ejemplo, la siguiente etiqueta <META>:

  <META HTTP-EQUIV="Expires" CONTENT="Tue, 20 Aug 1996 14:25:27 GMT">

  se convierte en la siguiente cabecera **HTTP**:

  Expires: Tue, 20 Aug 1996 14:25:27 GMT

  que puede ser usada por la cache del navegador o por los servidores proxy para saber hasta cuando se puede emplear la copia de una página existente en la caché.

- <META NAME="propiedad" CONTENT="contenido">. Se emplea para incluir propiedades del documento, tales como autor, fecha de caducidad, una lista de palabras clave, etc. El atributo NAME indica el nombre de la propiedad mientras que el atributo CONTENT especifica su valor. Por ejemplo, las siguientes etiquetas <META> indican el autor, una descripción, una lista de palabras clave y la fecha de creación de una página:

Ejemplo 3.3

```
<META NAME="Author" CONTENT="Sergio Luján Mora">
<META NAME="Rights" CONTENT="Sergio Luján Mora">
<META NAME="Description" CONTENT="Una página de prueba">
<META NAME="Keywords" CONTENT="curso, html, diseño, web">
<META NAME="Date" CONTENT="Monday, 1 January, 2001">
```

A título de ejemplo, la página principal de la web de la Universidad de Alicante contiene los siguientes metadatos:

Ejemplo 3.4

```
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
<META NAME="Author" CONTENT="Universidad de Alicante">
<META NAME="Copyright" CONTENT=" © Universidad de Alicante">
<META NAME="Description" CONTENT="Web que recoge toda la ...">
<META NAME="keywords" CONTENT="Universidad de Alicante, Alicante ...">
<META NAME="Formatter" CONTENT="Mozilla/4.5 [es](Win98; I)[Netscape]">
<META NAME="Generator" CONTENT="Mozilla/4.5 [es](Win98; I)[Netscape]">
```

```
<META NAME="robots" CONTENT="index, follow">
<META HTTP-EQUIV="Content-Language" CONTENT="ES">
<META HTTP-EQUIV="Content-Script-Type" CONTENT="JavaScript">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859">
```

3.7. Etiquetas HTML

Existen cientos de etiquetas, cada una con su conjunto de atributos. Además, no existe un estándar que acepten todos los navegadores: los dos principales navegadores, Netscape Navigator y Microsoft Internet Explorer, presentan diferencias entre ellos, aceptando cada uno etiquetas que no acepta el otro. Existe un intento de estandarización por parte de **W3C**, que periódicamente publica borradores, recomendaciones y estándares definitivos. La última recomendación es **XHTML** 1.0 (combinación de **HTML** y **XML**), que sustituye a **HTML** 4.01.

Las etiquetas **HTML** (reconocidas por Netscape Navigator 4.0 y superiores) las podemos clasificar en las siguientes categorías (en esta clasificación una etiqueta sólo aparece en una categoría, pero realmente hay etiquetas que se pueden clasificar en varias categorías):

- Etiquetas que definen la estructura del documento: <HTML>, <HEAD> y <BODY>.
- Etiquetas que pueden ir en la cabecera: <TITLE>, <BASE>, <META>, <STYLE> y <LINK>.
- Etiquetas que definen bloques de texto: <ADDRESS>, <BLOCKQUOTE>, <DIV>, <H1> ... <H6>, <P>, <PRE> y <XMP>.
- Etiquetas de listas: <DIR>, <DL>, <DT>, <DD>, <MENU>, <OL>, <UL> y <LI>.
- Etiquetas de características del texto: <B>, <BASEFONT>, <BIG>, <BLINK>, <CITE>, <CODE>, <EM>, <FONT>, <I>, <KBD>, <PLAINTEXT>, <SMALL>, <S>, <STRIKE>, <STRONG>, <SUB>, <SUP>, <TT>, <U> y <VAR>.
- Etiquetas de anclas y enlaces: <A>.
- Etiquetas de imágenes y mapas de imágenes: <IMG>, <AREA> y <MAP>.
- Etiquetas de tablas: <TABLE>, <CAPTION>, <TR>, <TD> y <TH>.
- Etiquetas de formularios: <FORM>, <INPUT>, <SELECT>, <OPTION>, <TEXTAREA>, <KEYGEN> y <ISINDEX>.

- Etiquetas de marcos: <FRAME>,<FRAMESET> y <NOFRAMES>.
- Etiquetas de situación de contenidos: <LAYER>, <ILAYER> y <NOLAYER>.
- Etiquetas de script: <SCRIPT>, <NOSCRIPT> y <SERVER>.
- Etiquetas de applets y plug-ins: <APPLET>, <PARAM>, <EMBED>, <NOEMBED> y <OBJECT>.
- Etiquetas de ajuste del texto:
, <CENTER>, <HR>, <MULTICOL>, <NOBR>, <SPACER>, <SPAN> y <WBR>.

Vamos a comentar las etiquetas más importantes (las más empleadas) con los atributos más comunes. No es una explicación exhaustiva, pero suficiente para un primer contacto con **HTML**. En el Apéndice A se puede consultar una relación con todas las etiquetas que acepta Netscape Communicator 4.0.

3.8. Formato del texto

En esta sección vamos a repasar las etiquetas más importantes que permiten asignar formato al texto: los encabezados de sección, los estilos lógicos y físicos, la etiqueta <FONT> ... </FONT> y por último, como alinear el texto.

3.8.1. Encabezados de secciones

Existen seis niveles de encabezados, numerados del 1 al 6, y según tamaños decrecientes: el nivel 1 es la etiqueta <H1> ... </H1> (la más prominente) y el nivel 6 es la etiqueta <H6> ... </H6> (la menos prominente). Los encabezados se suelen mostrar con tipos de letra más grandes, en negrita o más enfatizados que el texto normal. Pero los niveles 5 y 6 aparecen normalmente con un tamaño inferior al del texto normal. El tamaño de cada encabezado lo selecciona el navegador, por lo que la apariencia puede variar significativamente de uno a otro.

Cuando se visualizan, los encabezados comienzan en una línea nueva y se suele dejar un espacio en blanco extra antes de ellos. La sintaxis de esta etiqueta es:

Ejemplo 3.5

```
<Hn ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"> ... </Hn>
```

donde n es un número del 1 al 6 y el atributo ALIGN especifica el alineamiento horizontal del encabezado‹ 21 ›. Por ejemplo, en la Figura 3.2 vemos como se visualiza el siguiente código en un navegador.

Ejemplo 3.6

```
<HTML>
<BODY>
Texto normal
<H1>Encabezado nivel 1</H1>
<H2>Encabezado nivel 2</H2>
<H3>Encabezado nivel 3</H3>
<H4>Encabezado nivel 4</H4>
<H5>Encabezado nivel 5</H5>
<H6>Encabezado nivel 6</H6>
Texto normal
</BODY>
</HTML>
```

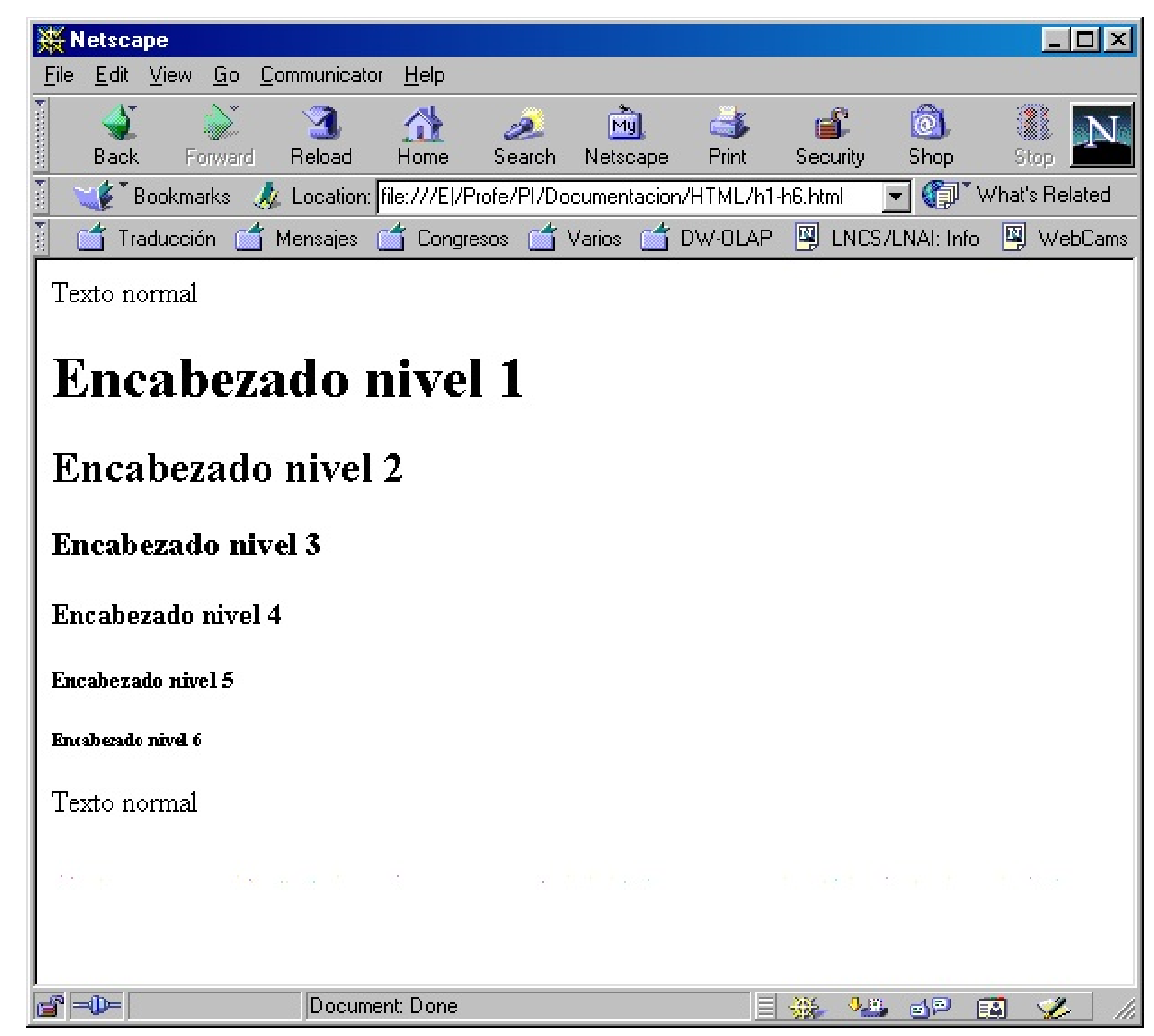


Figura 3.2: Ejemplo de encabezados

3.8.2. Formatos de caracteres

Las siguientes etiquetas se pueden emplear en línea (*inline*), lo que significa que no interrumpen el fluir del texto (no producen saltos de línea ni nada similar). Por tanto, se pueden aplicar a caracteres individuales.

Los formatos (también llamados estilos) se dividen en lógicos y físicos. Los formatos lógicos dependen del navegador (cada uno lo puede interpretar de distinta forma), mientras que los formatos físicos siempre se representan de la misma forma.

El siguiente código **HTML** muestra las etiquetas de los formatos más comunes. En la Figura 3.3 vemos como se visualiza en un navegador.

Ejemplo 3.7

```
<HTML>
<BODY>
Formatos físicos:<BR>
<B>Texto en negrita: <B></B><BR>
<I>Texto en cursiva: <I></I><BR>
<U>Texto subrayado: <U></U><BR>
<TT>Texto en teletipo (fuente fija): <TT></TT><BR>
<STRIKE>Texto tachado: <STRIKE></STRIKE><BR>
<S>También funciona <S></S><BR>
<BR>
Formatos lógicos:<BR>
<CITE>Cita: <CITE></CITE><BR>
<CODE>Código: <CODE></CODE><BR>
<DFN>Definición: <DFN></DFN><BR>
<EM>Enfatizado: <EM></EM><BR>
<KBD>Texto tecleado: <KBD></KBD><BR>
<STRONG>Texto grueso: <STRONG></STRONG><BR>
<VAR>Texto variable: <VAR></VAR><BR>
</BODY>
</HTML>
```

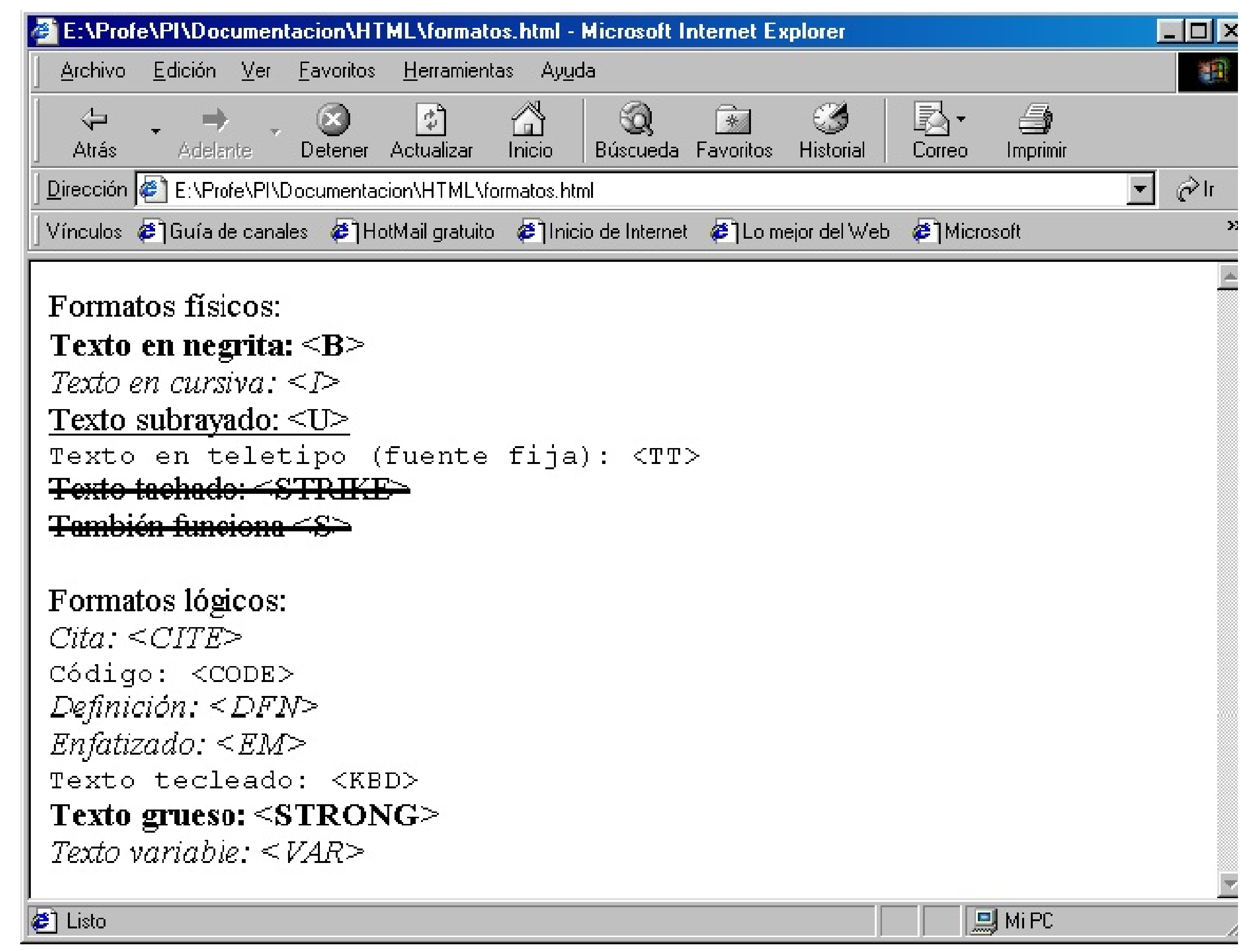


Figura 3.3: Formatos físicos y lógicos

3.8.3. La etiqueta <FONT>

La etiqueta <FONT> ... </FONT> permite modificar el tipo de letra, el tamaño y el color del texto . Todo el texto entre las etiquetas <FONT> ... </FONT> se muestra según los valores especificados en los atributos de la etiqueta.

Para modificar el tipo de letra se emplea el atributo FACE. Se le puede indicar una lista de tipos de letras separados por comas. El navegador comprueba si el primer tipo de letra está disponible, si no es así prueba con el segundo y así sucesivamente. Si ninguno de los tipos de letra está disponible, no se produce ningún cambio en el tipo de letra.

Los tipos de letra pueden ser específicos o genéricos. Entre los primeros, los más empleados son Arial, Courier, Helvetica, Tahoma, Times y Verdana. Los tipos de letra genéricos son serif, sans-serif, cursivev, monospace y fantasy. El siguiente código muestra el uso de distintos tipos de letra. En la Figura 3.4 podemos observar como se visualiza en un navegador. Como se ve en la imagen del navegador, el texto escrito en Helvetica

aparece con el tipo de letra por defecto; esto se debe a que en el ordenador en el que se visualiza la página no está disponible el tipo de letra Helvetica.

Ejemplo 3.8

```
<HTML>
<BODY>
Tipos de letra específicos:<BR>
<FONT FACE="Arial">Texto en Arial</FONT><BR>
<FONT FACE="Helvetica">Texto en Helvetica</FONT><BR>
<FONT FACE="Tahoma">Texto en Tahoma</FONT><BR>
<BR>
Tipos de letra genéricos:<BR>
<FONT FACE="serif">Texto en serif</FONT><BR>
<FONT FACE="sans-serif">Texto en sans-serif</FONT><BR>
<FONT FACE="cursive">Texto en cursive</FONT><BR>
<FONT FACE="monospace">Texto en monospace</FONT><BR>
<FONT FACE="fantasy">Texto en fantasy</FONT><BR>
</BODY>
</HTML>
```

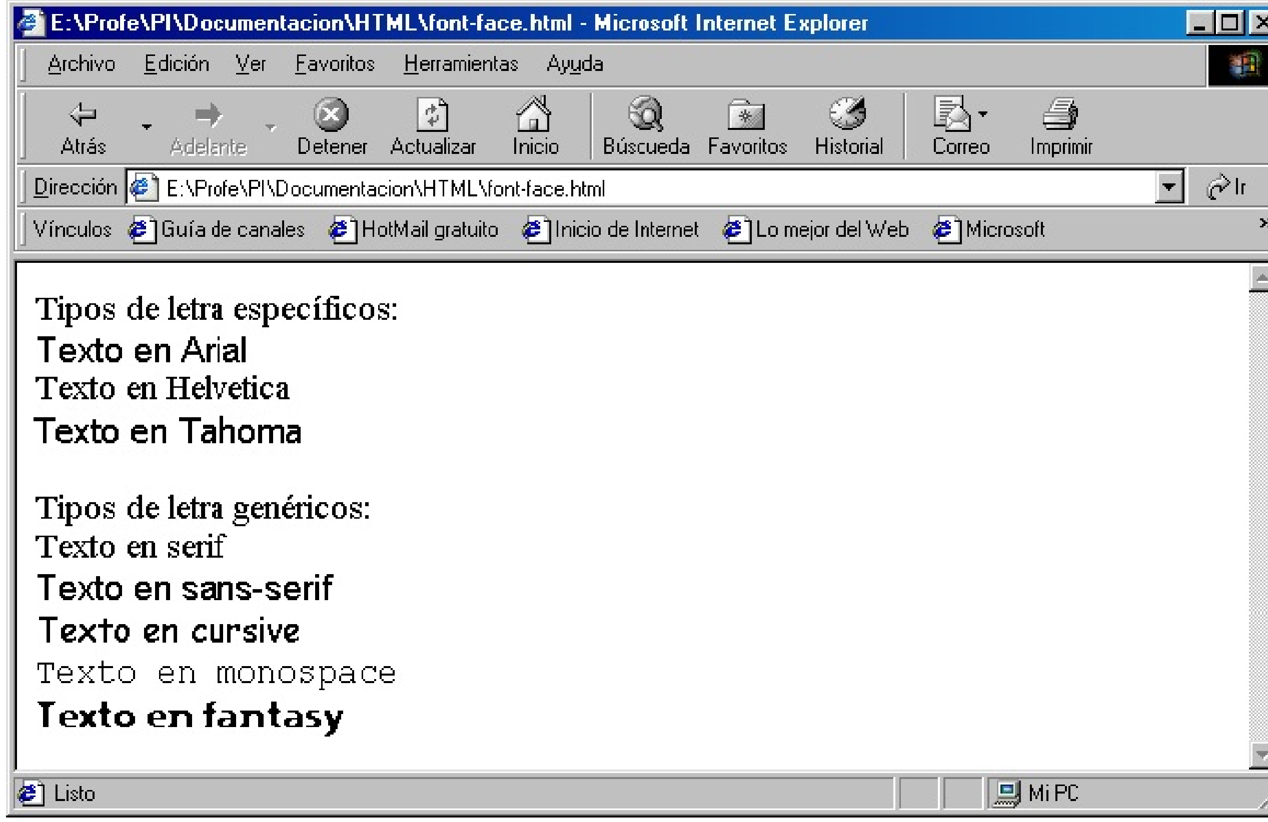


Figura 3.4: Distintos tipos de letra con la etiqueta <FONT>

Para modificar el tamaño se emplea el atributo SIZE. Este atributo define el tamaño de forma relativa, en un intervalo del 1 (letra más pequeña) a 7 (letra más grande). El tamaño base por defecto es 3‹ 22 ›. También se puede indicar un valor relativo al tamaño base si se emplean los signos + o -. Por ejemplo, +2 significa un

incremento en dos unidades respecto al tamaño base. En la Figura 3.5 vemos como se visualiza el siguiente código que muestra el uso del atributo SIZE.

Ejemplo 3.9

```
<HTML>
<BODY>
Distintos tamaños de letra:<BR>
<FONT SIZE="1">Texto en 1</FONT><BR>
<FONT SIZE="2">Texto en 2</FONT><BR>
<FONT SIZE="3">Texto en 3</FONT><BR>
<FONT SIZE="4">Texto en 4</FONT><BR>
<FONT SIZE="5">Texto en 5</FONT><BR>
<FONT SIZE="6">Texto en 6</FONT><BR>
<FONT SIZE="7">Texto en 7</FONT><BR>
</BODY>
</HTML>
```

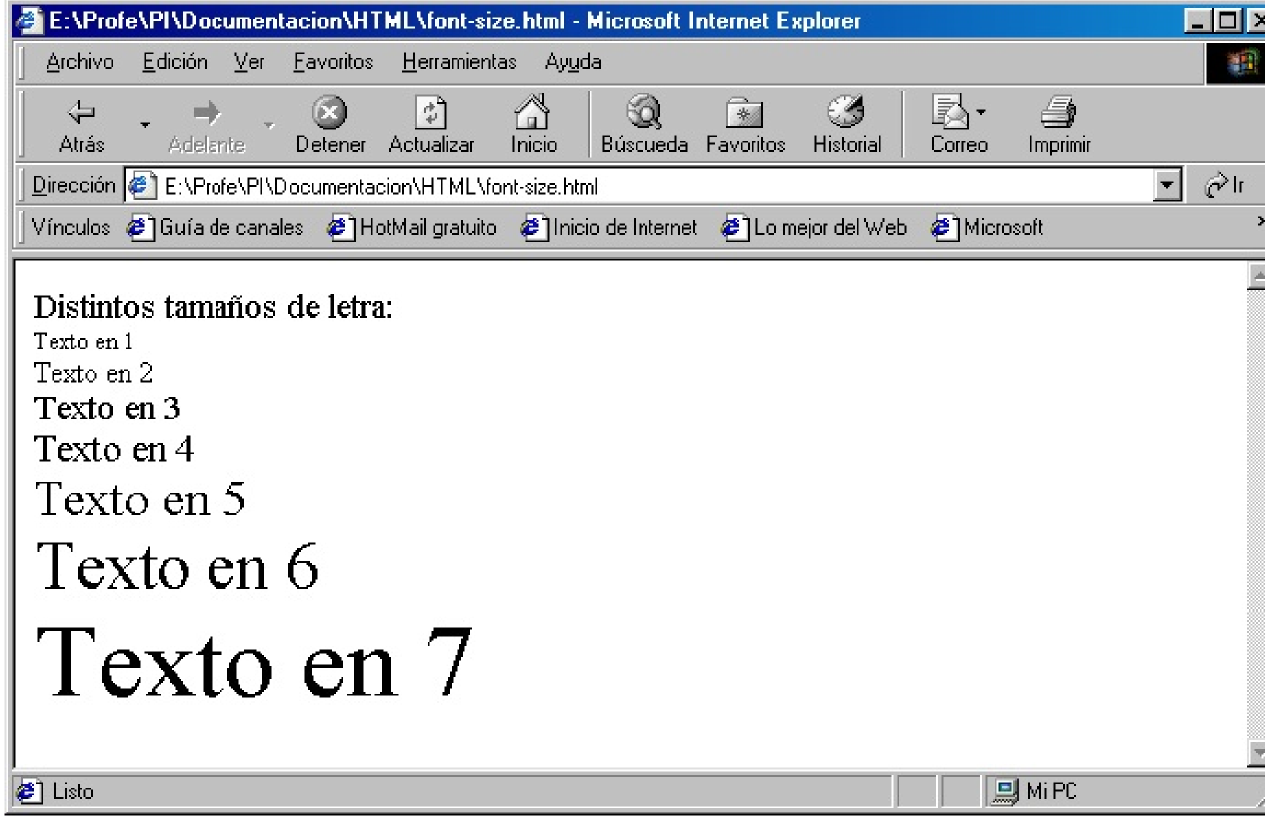


Figura 3.5: Distintos tamaños de letra con la etiqueta <FONT>

3.8.4. Alineamiento del texto

La etiqueta <P> ... </P> se emplea para marcar párrafo de texto. Un párrafo comienza en una línea nueva y el navegador suele dejar un espacio en blanco extra

antes del párrafo.

Para alinear el contenido de un párrafo se emplea el atributo ALIGN. El contenido del párrafo puede alinearse a la izquierda (LEFT), a la derecha (RIGHT), centrado (CENTER) o justificado (JUSTIFY). El siguiente código **HTML** muestra el mismo párrafo alineado de cuatro formas distintas. En la Figura 3.6 podemos ver el código visualizado en un navegador.

Ejemplo 3.10

```
<HTML>
<BODY>
<P ALIGN="LEFT">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="RIGHT">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="CENTER">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="JUSTIFY">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
</BODY>
</HTML>
```

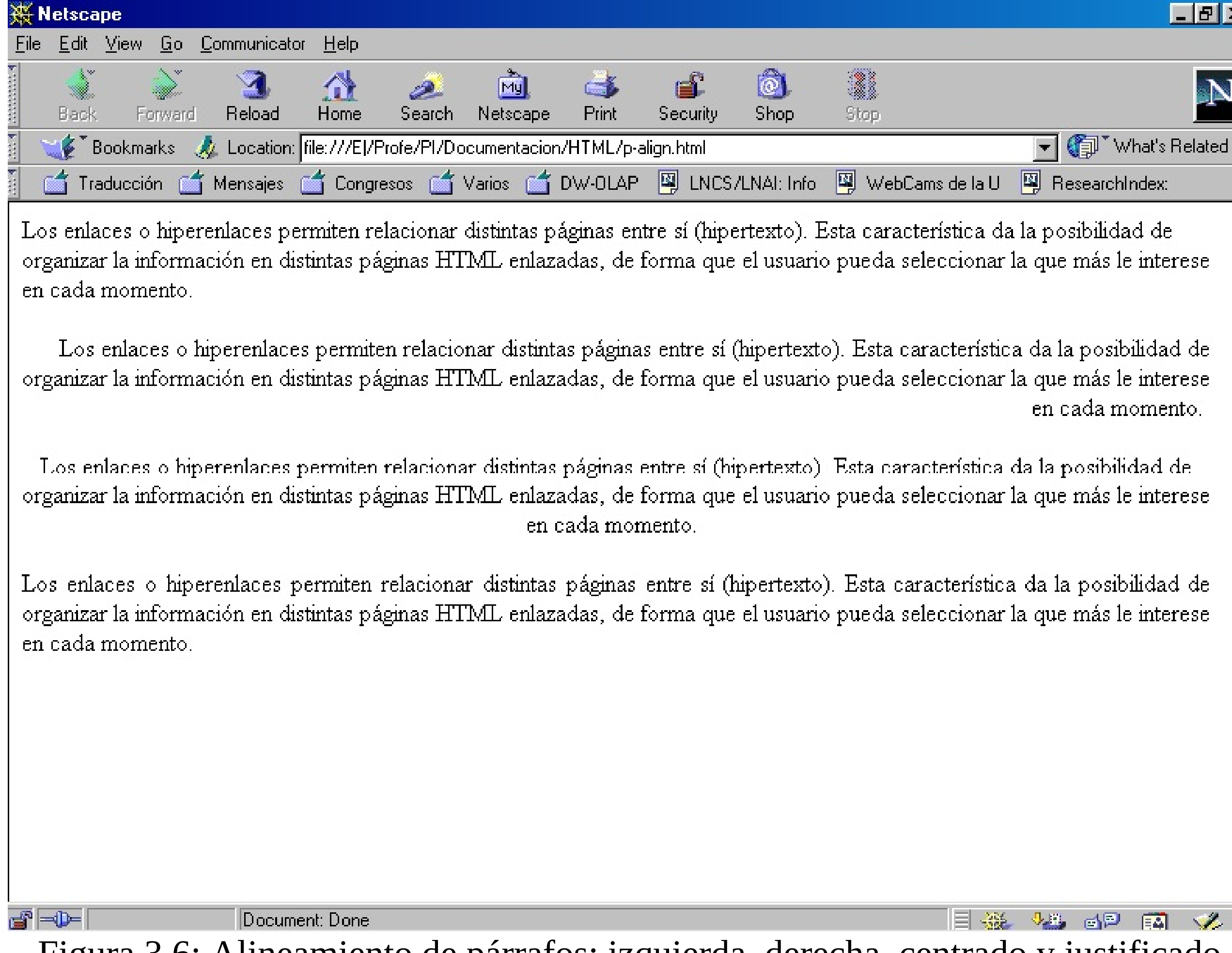


Figura 3.6: Alineamiento de párrafos: izquierda, derecha, centrado y justificado

Los bloques de texto se especifican usando la etiqueta <BLOCKQUOTE> ... </BLOCKQUOTE>. Un bloque de texto aparece sangrado hacia la derecha. Se suele emplear para marcar citas textuales o deniciones de especial relevancia. Los bloques de texto se pueden anidar para producir un mayor sangrado. El siguiente código muestra el empleo de esta etiqueta. En la Figura 3.7 se puede ver como se muestra en un navegador web.

Ejemplo 3.11

```
<HTML>
<BODY>
Este texto no tiene sangría.
<BLOCKQUOTE>
1. Un nivel de sangría.
<BLOCKQUOTE>
2. Dos niveles de sangría.
<BLOCKQUOTE>
3. Tres niveles de sangría.
</BLOCKQUOTE>
</BLOCKQUOTE>
Volvemos al nivel 1.
</BLOCKQUOTE>
Volvemos a texto
```

```
sin sangría.
</BODY>
</HTML>
```

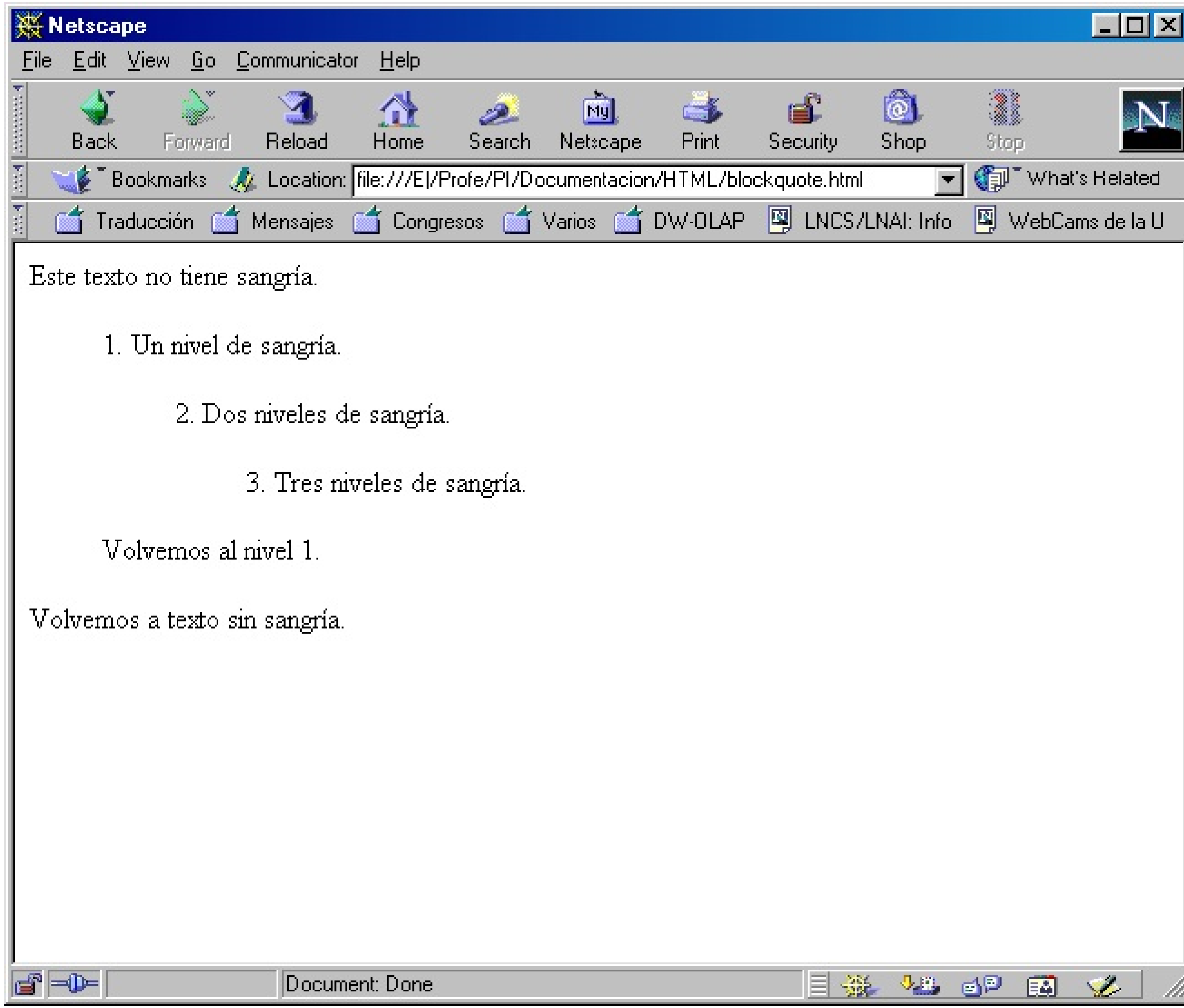


Figura 3.7: Bloques de texto con distinta sangría

3.9. Listas

Las listas permiten organizar la información de una manera lógica, lo que facilita su legibilidad. Existen cinco tipos de listas en **HTML**: listas de definición, listas ordenadas, listas no ordenadas, listas de directorio y listas de menú. Como las dos últimas listas están obsoletas, ya que se visualizan como las listas no ordenadas, no las vamos a ver.

Las listas se pueden anidar entre sí, incluso si son de distinto tipo. En el caso de anidar listas no numeradas, cada nivel de anidamiento tendrá un tipo de símbolo distinto.

3.9.1. Listas de definición

Una lista de definición se emplea para mostrar términos con sus correspondientes definiciones, como si se tratase de un glosario o diccionario.

Una lista de definición se crea con la etiqueta <DL> ... </DL> (*definition list*). Contiene una serie de términos, que se definen con la etiqueta <DT> (*definition term*), y cada término posee una o más definiciones, que se indican cada una con la etiqueta <DD> (*definition description*). Las definiciones de cada término aparecen con una sangría hacia la derecha.

El siguiente ejemplo muestra como se usan estas etiquetas. Aunque las líneas que contienen las etiquetas <DD> aparecen con unos espacios en blanco al principio, estos no influyen para nada en su visualización. En la Figura 3.8 vemos el resultado que produce este ejemplo.

Ejemplo 3.12

```
<HTML>
<BODY>
<DL>
<DT>BANCO
<DD>Lugar donde se deposita dinero
<DD>Sitio donde se sienta la gente
<DT>GATO
<DD>Animal de cuatro patas con pelo
<DD>Herramienta para levantar un vehículo
<DT>ORDENADOR
<DD>Aparato electrónico que realiza cálculos
</DL>
</BODY>
</HTML>
```

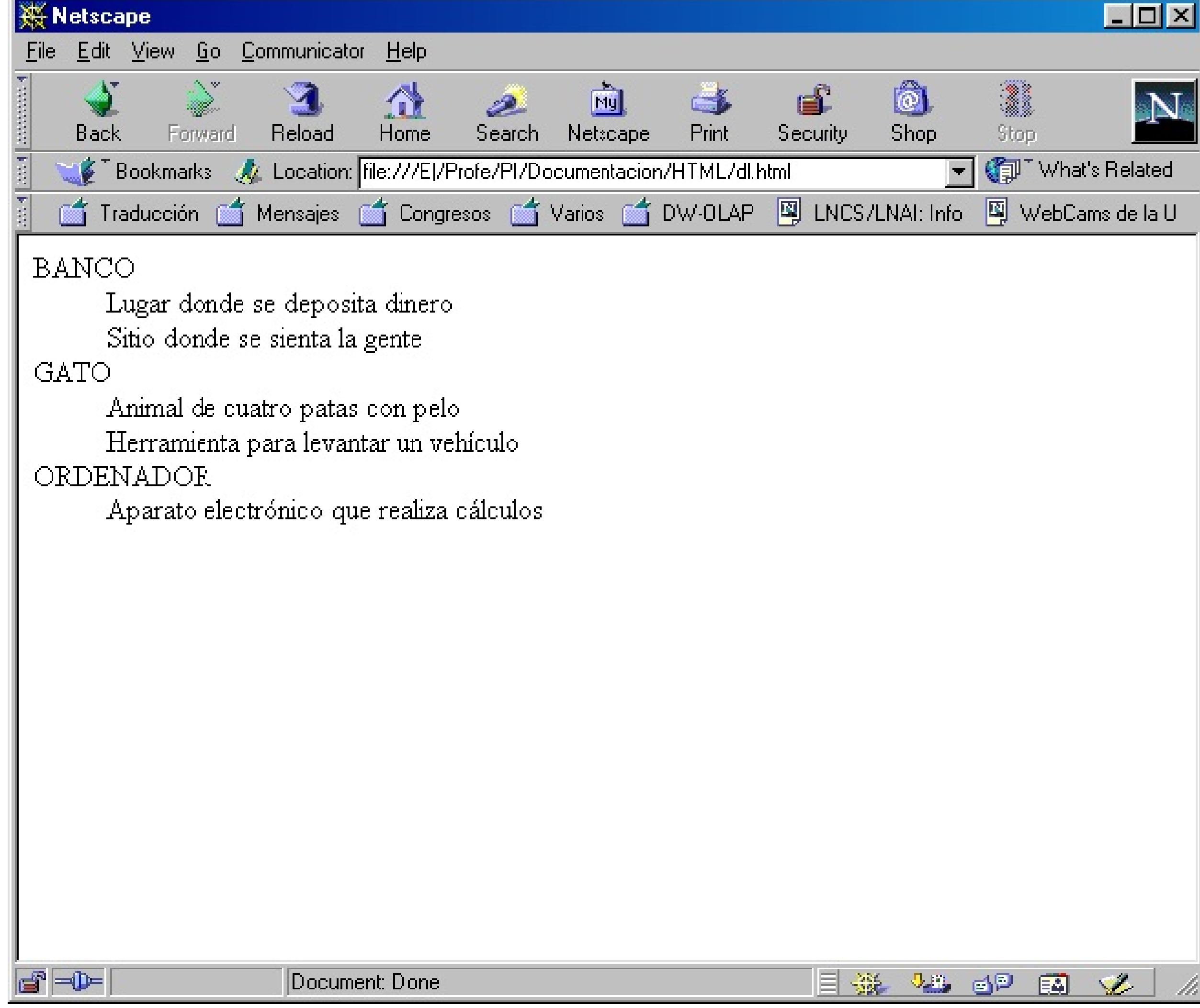


Figura 3.8: Listas de definición

3.9.2. Listas ordenadas

En las listas ordenadas o numeradas, cada elemento aparece numerado. La etiqueta <OL> ... <OL> (*ordered list*) define una lista de este tipo. Cada elemento se define con la etiqueta <LI> (*list item*). Esta etiqueta posee dos atributos: START y TYPE.

El atributo START indica el valor desde el que se empieza la numeración; tiene que ser un valor positivo.

El atributo TYPE indica el tipo de numeración de los elementos de la lista. Los posibles valores de este atributos son:

- A: indica una numeración con letras en mayúscula.
- a: indica una numeración con letras en minúscula.
- I: indica una numeración con números romanos en mayúsculas.

- i: indica una numeración con números romanos en minúsculas.
- 1: indica una numeración con números.

El siguiente ejemplo muestra el uso de esta etiqueta. Además, también se puede ver como se pueden anidar listas (incluir una lista dentro de otra lista). En la Figura 3.9 vemos como se muestra esta página.

Ejemplo 3.13

```
<HTML>
<BODY>
Lista normal, con anidamiento:
<OL>
<LI>Elemento 1
<LI>Elemento 2
<LI>Lista anidada:
        <OL>
        <LI>Elemento 1
        <LI>Elemento 2
        </OL>
</OL>
Lista numerada con letras en mayúsculas:
<OL TYPE="A">
<LI>Elemento 1
<LI>Elemento 2
</OL>
Lista numerada con números romanos:
<OL TYPE="i">
<LI>Elemento 1
<LI>Elemento 2
</OL>
</BODY>
</HTML>
```

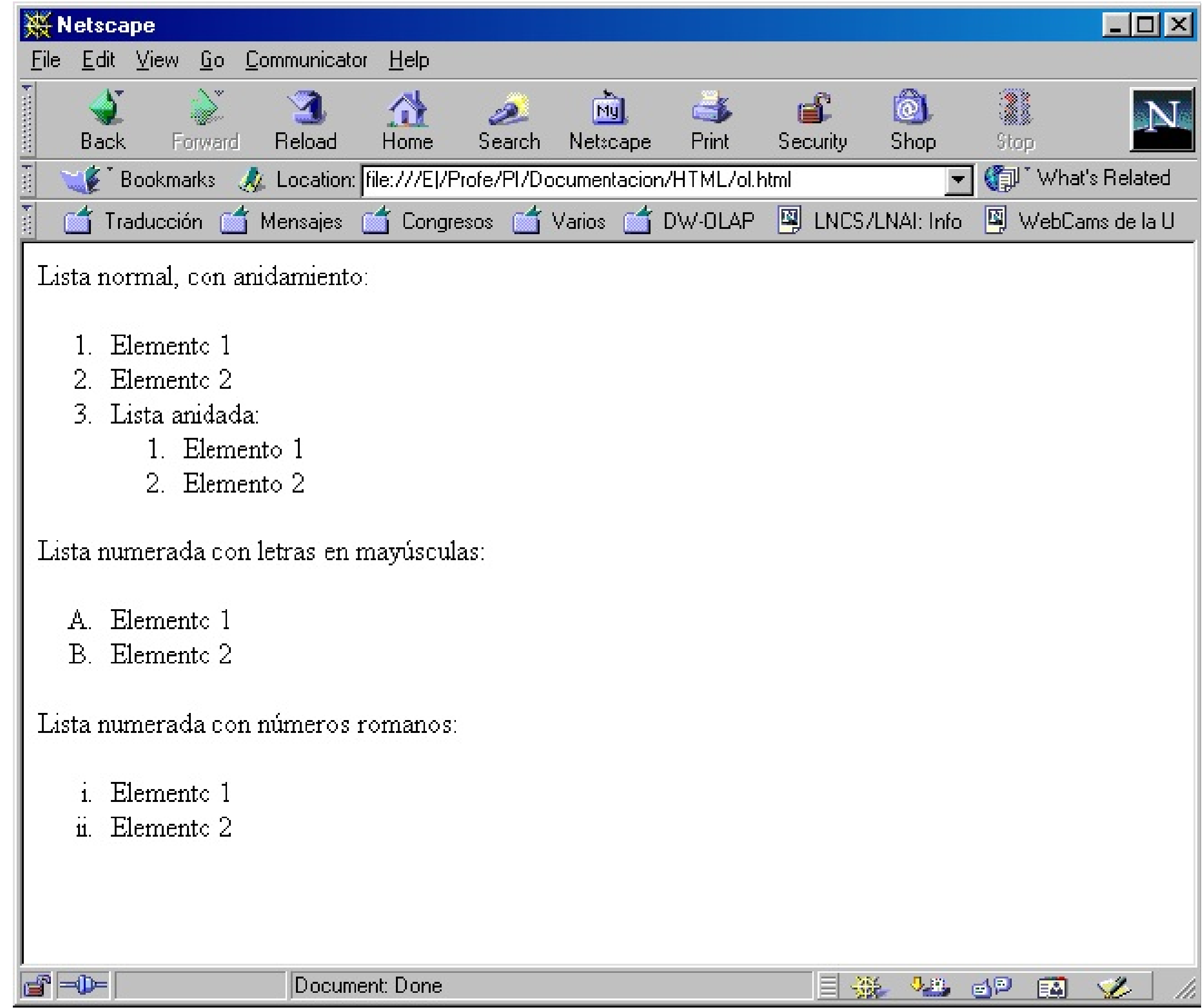


Figura 3.9: Listas ordenadas

3.9.3. Listas no ordenadas

En las listas no ordenadas, los elementos aparecen marcados mediante unos pequeños elementos gráficos, llamados en inglés *bullet*. La etiqueta <UL> ... </UL> (*unordered list*) define una lista no ordenada. Cada elemento se define con la etiqueta <LI> (*list item*).

Esta etiqueta posee el atributo TYPE, que permite cambiar el elemento gráfico empleado para marcar los elementos. Los posibles valores que puede tomar este atributo son:

- CIRCLE: un disco con el centro vacío.
- DISC: un disco sólido.
- SQUARE: un cuadrado.

El siguiente código muestra como se emplea esta etiqueta. Además, en la última lista hay un anidamiento de varias listas. En la Figura 3.10 se puede ver que cuando se anidan varias listas, el elemento gráfico *bullet* cambia automáticamente, según cual sea el nivel de anidamiento.

Ejemplo 3.14

```
<HTML>
<BODY>
<UL TYPE="CIRCLE">
<LI>Elemento 1
<LI>Elemento 2
</UL>
<UL TYPE="DISC">
<LI>Elemento 1
<LI>Elemento 2
</UL>
<UL TYPE="SQUARE">
<LI>Elemento 1
<LI>Elemento 2
</UL>
<UL>
<LI>Elemento a:
        <UL>
        <LI>Elemento b:
                <UL>
                <LI>Elemento c:
                </UL>
        </UL>
</UL>
</BODY>
</HTML>
```

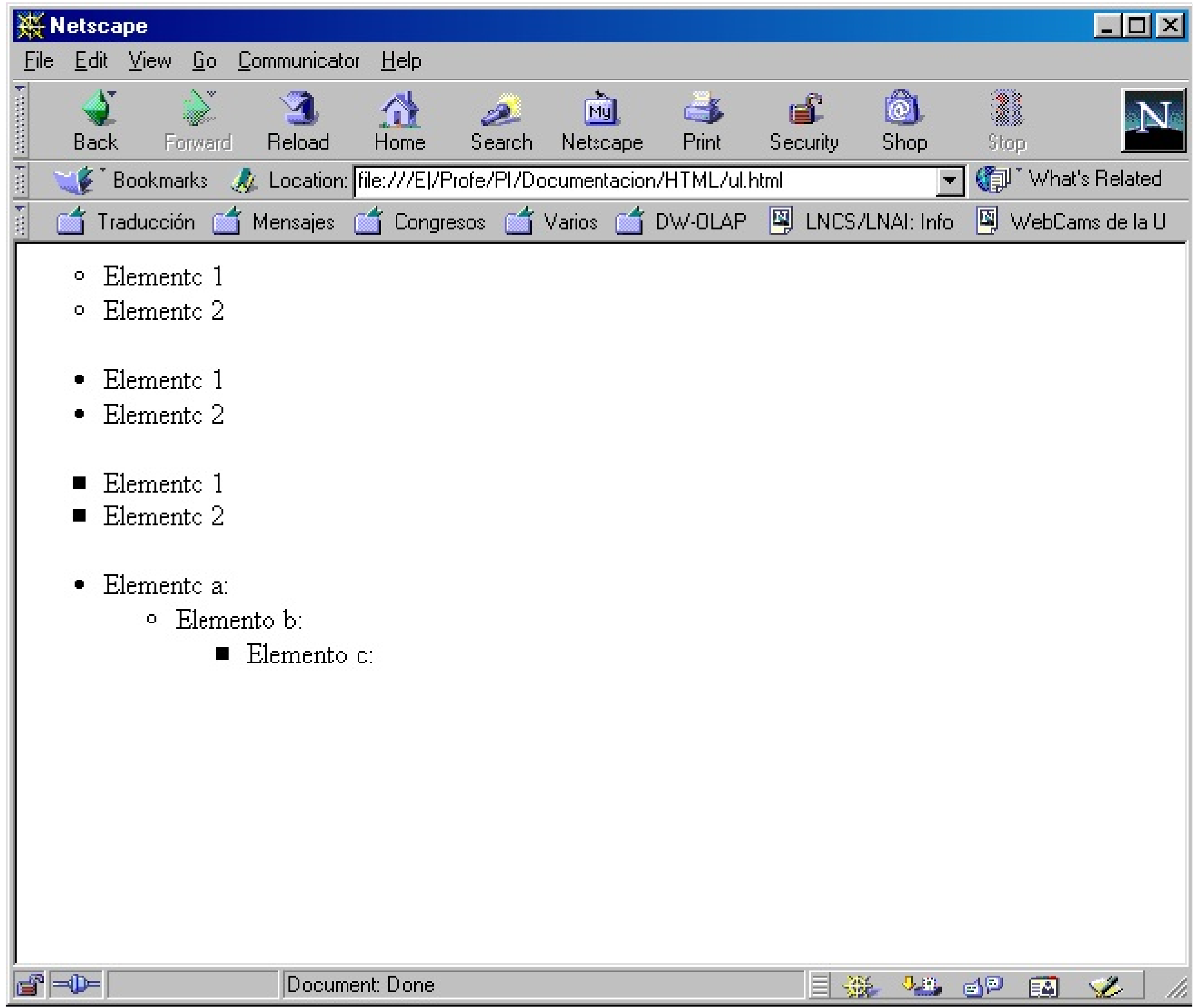


Figura 3.10: Listas no ordenadas

3.10. Colores

En HTML se puede cambiar el color de distintos elementos, como el color de fondo de una página, el color del texto, el color de una tabla, etc. Existen dos formas de especificar los colores:

- Mediante las componentes *Red Green Blue* (**RGB**). Mediante esta codificación, se especifica la intensidad de los colores básicos rojo, verde y azul que permiten obtener por combinación cualquier otro color. Cada componente puede variar entre 0 y 255, por lo que se tienen 16 777 216 (256 x 256 x 256) colores. Las componentes se tienen que expresar en hexadecimal y al principio se tiene que poner el signo #; por tanto, el formato general es #RRGGBB. Por ejemplo, el color negro se representa mediante #000000, el color blanco como #FFFFFF y un rojo brillante como #FF0000.

- Mediante los nombres de los colores. Existen una serie de colores a los que se les ha asignado un nombre en inglés, como *green*, *olive* o *white*.

En el Apéndice B se incluye más información sobre el uso de colores en **HTML** y como pasar las componentes **RGB** de decimal a hexadecimal.

3.10.1. Color de fondo de una página

El color de fondo de una página se puede cambiar mediante el atributo BGCOLOR de la etiqueta <BODY> ... </BODY>.

3.10.2. Color del texto

La etiqueta <FONT> ... </FONT> posee el atributo COLOR que permite indicar el color del texto. También se puede cambiar el color del texto para toda una página, el color de los enlaces, el color de los enlaces visitados y el color de los enlaces al activarse. Para ello se emplean los atributos TEXT, LINK, VLINK y ALINK de la etiqueta <BODY> ... </BODY>.

En el siguiente ejemplo, se cambia el color del texto y de los enlaces para que todos tengan el mismo color y no se puedan diferenciar unos de otros.

Ejemplo 3.15

```
<HTML>
<BODY TEXT="black" LINK="black" VLINK="text" ALINK="black">
<A HREF="http://www.ua.es">Universidad de Alicante</A>
<BR>
<A HREF="http://www.eps.ua.es">Escuela Politécnica Superior</A>
</BODY>
</HTML>
```

3.11. Enlaces

Los enlaces o hiperenlaces permiten relacionar distintas páginas entre sí (hipertexto). Esta característica da la posibilidad de organizar la información en distintas páginas **HTML** enlazadas, de forma que el usuario pueda seleccionar la que más le interese en cada momento.

Un hiperenlace puede hacer referencia a un punto determinado de la página que lo contiene, a otra página **HTML** o a un punto determinado de otra página **HTML**. En los dos últimos casos, la página destino puede residir en el mismo servidor que la página que contiene el hiperenlace o en otro distinto.

La etiqueta que utiliza **HTML** para definir un hiperenlace es `<A> ... </A>`. Todo aquello que aparezca en una página **HTML** entre dichas etiquetas se considera un hiperenlace (será el objeto sensible que al ser pulsado producirá el salto al destino del enlace). Normalmente suele utilizarse texto e imágenes como hiperenlaces.

3.11.1. Enlace a un punto del mismo documento

Un enlace de este tipo consta de dos partes: una referencia y un destino. El destino se identifica por un nombre. La referencia hará alusión al nombre del destino. En una página se pueden incluir varias referencias a un mismo destino, pero no se pueden crear varios destinos con el mismo nombre. La forma de definir este enlace es:

- Referencia: `<A HREF="#nombre">objeto del enlace</A>`.
- Destino: `<A NAME="nombre">objeto del destino</A>`.

Como puede observarse, en el caso de la referencia el nombre al que hace alusión va precedido del símbolo almohadilla (#), mientras que en el destino no.

En el siguiente código de ejemplo, tenemos un enlace sobre la misma página. En la Figura 3.11 vemos que la palabra enlace es un enlace; al pulsar sobre ella saltamos al destino que podemos ver en la Figura 3.12.

Ejemplo 3.16

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
Aquí tenemos un <A HREF="#destino">enlace</A> sobre la misma
página. Hay que dejar muchas líneas en blanco para que se
pueda comprobar el efecto.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<A NAME="destino">Esto</A> es el destino del enlace que aparece
al principio de la página.
</BODY>
</HTML>
```

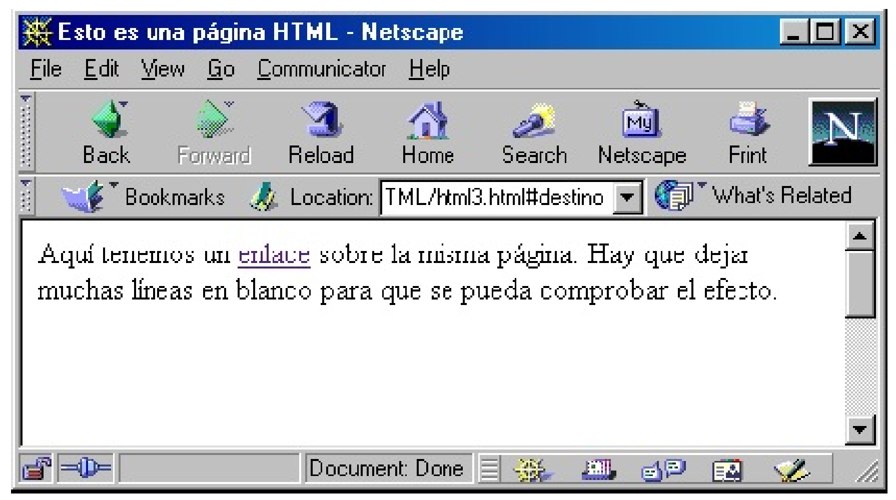


Figura 3.11: Enlace a un destino interno



Figura 3.12: Destino del enlace interno

3.11.2. Enlace a otro documento

Para incluir un enlace en un documento a otro documento, simplemente hay que incluir en el documento origen una referencia al documento destino. En este último no hace falta indicar que es destino de un enlace. La forma de definir este enlace es:

- Referencia: <A HREF="pagina.html">objeto del enlace</A>.

Muy importante: cuando indiquemos el nombre del documento destino (pagina.html), hay que tener mucho cuidado con las mayúsculas y minúsculas. En el siguiente ejemplo tenemos dos páginas con colores de fondo distintos; en cada una de ellas figura un enlace a la otra.

- Página html4a.html (Figura 3.13):

Ejemplo 3.17

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY BGCOLOR="#FFBBBB">
<B>Aquí tenemos un <A HREF="html4b.html">enlace</A> a otra
página que tiene el fondo azul.</B>
</BODY>
</HTML>
```

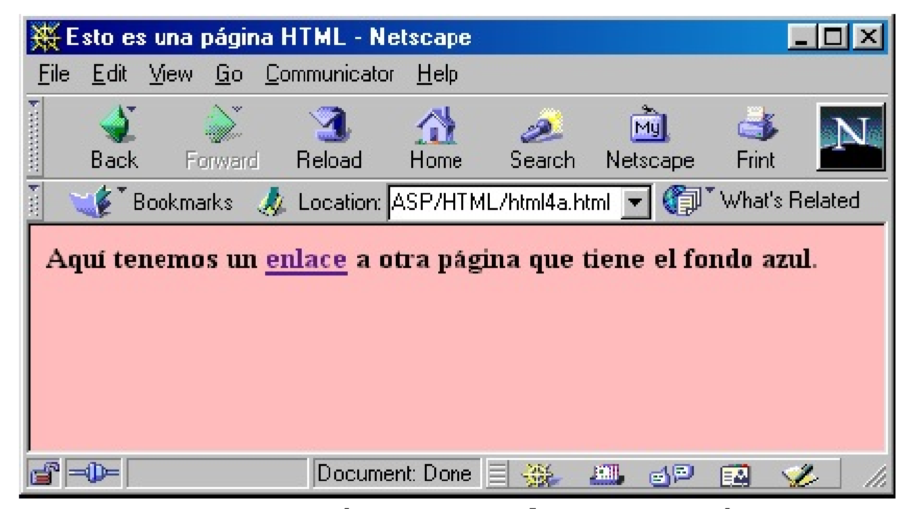


Figura 3.13: Página con enlace a otra página

- Página html4b.html (Figura 3.14):

Ejemplo 3.18

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY BGCOLOR="#BBBBFF">
<B>Aquí tenemos un <A HREF="html4a.html">enlace</A> a otra
página que tiene el fondo rojo.</B>
</BODY>
</HTML>
```



Figura 3.14: Página con enlace a otra página

3.11.3. Enlace a un punto de otro documento

Este tipo de enlace es una combinación de los dos anteriores. La forma de definir este enlace es:

- Referencia: <A HREF="pagina.html#nombre">objeto del enlace</A>.
- Destino: <A NAME="nombre">objeto del destino</A>.

En el siguiente ejemplo tenemos una página con dos enlaces a la misma página, pero a distintos destinos dentro de esa página.

- Página html5a.html (Figura 3.15):

Ejemplo 3.19

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
<B>Aquí tenemos un <A HREF="html5b.html#destino1">enlace</A>
a un destino de otra página.
<BR><BR>
Aquí tenemos otro <A HREF="html5b.html#destino2">enlace</A>
a otro destino de la misma página que antes.
</B>
</BODY>
</HTML>
```

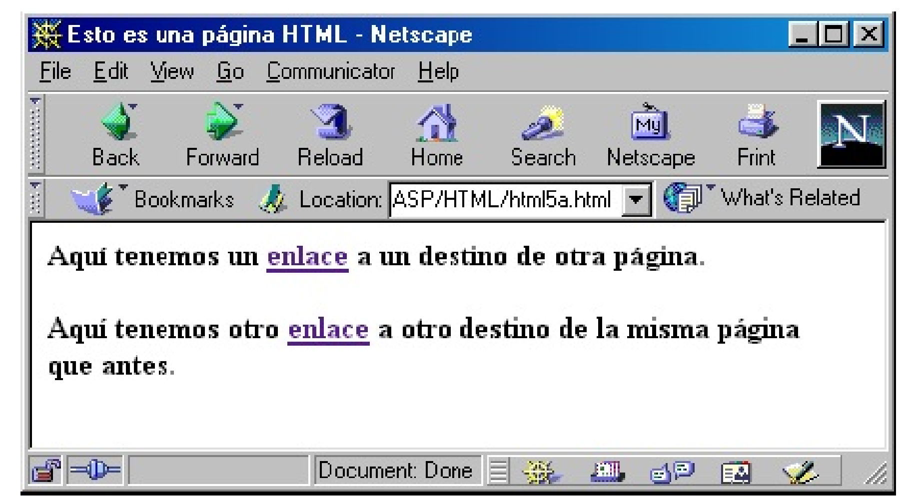


Figura 3.15: Página con dos enlaces a otra página

- Página html5b.html (Figura 3.16):

Ejemplo 3.20

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
<B>Aquí tenemos un <A NAME="destino1">destino</A>, el número 1.
<BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR><BR><BR><BR><BR>
Aquí tenemos otro <A NAME="destino2">destino</A>, el número 2.
</B>
</BODY>
</HTML>
```

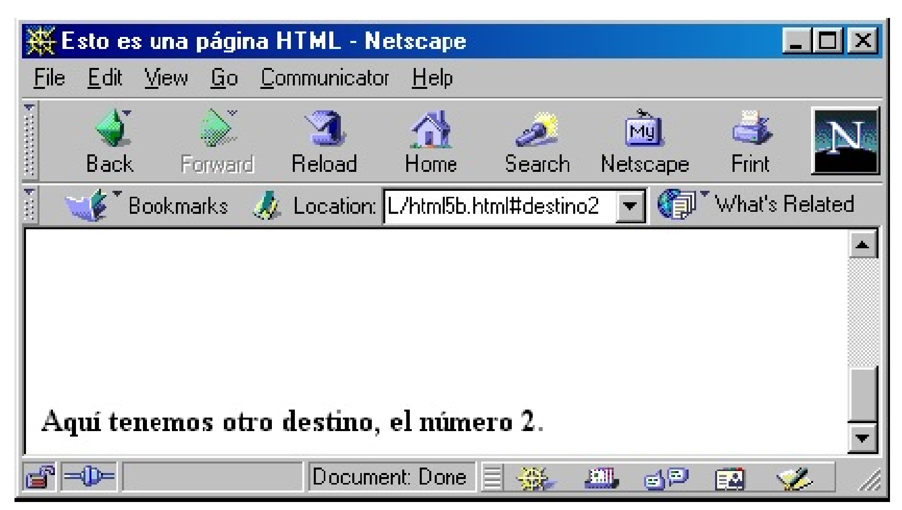


Figura 3.16: Página destino de los enlaces

3.12. Tablas

Hasta que aparecieron las tablas en el lenguaje **HTML**, la única posibilidad de tabular cosas en una página consistía en usar texto con preformato (<PRE> ... </PRE>) y colocar manualmente los espacios hasta que las columnas estuviesen perfectamente alineadas. Este proceso, además de ser realmente tedioso, no ofrece resultados con la vistosidad deseada.

Las tablas se construyen siguiendo una serie de reglas sencillas:

- Las tablas en **HTML** se definen mediante los códigos pareados <TABLE> ... </TABLE>.
- Una tabla se compone de celdillas de datos. Una celdilla se define usando los códigos <TD> ... </TD>.
- Las celdillas se agrupan en filas, que se definen con los códigos <TR> ... </TR>.
- Pueden existir celdillas que se emplean como encabezados de filas o columnas, y se definen con los códigos <TH> ... </TH>. Este tipo de celdas suele aparecer diferenciada de las otras, normalmente en negrita.
- Las celdillas pueden contener cualquier elemento **HTML**: texto, imágenes, enlaces e incluso otras tablas anidadas.

El siguiente ejemplo muestra una tabla con bordes formada por nueve celdas, de las que tres forman parte del encabezado. La tabla se ha dividido en tres filas y tres columna. En la Figura 3.17 vemos como se ve esta página en una navegador.

Ejemplo 3.21

```
<HTML>
<BODY>
<TABLE BORDER="1">
<TR>
	<TH>Cabecera 1</TH>
	<TH>Cabecera 2</TH>
	<TH>Cabecera 3</TH>
</TR>
<TR>
	<TD>Elemento (1, 1)</TD>
	<TD>Elemento (1, 2)</TD>
	<TD>Elemento (1, 3)</TD>
</TR>
<TR>
```

```
        <TD>Elemento (2, 1)</TD>
        <TD>Elemento (2, 2)</TD>
        <TD>Elemento (2, 3)</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

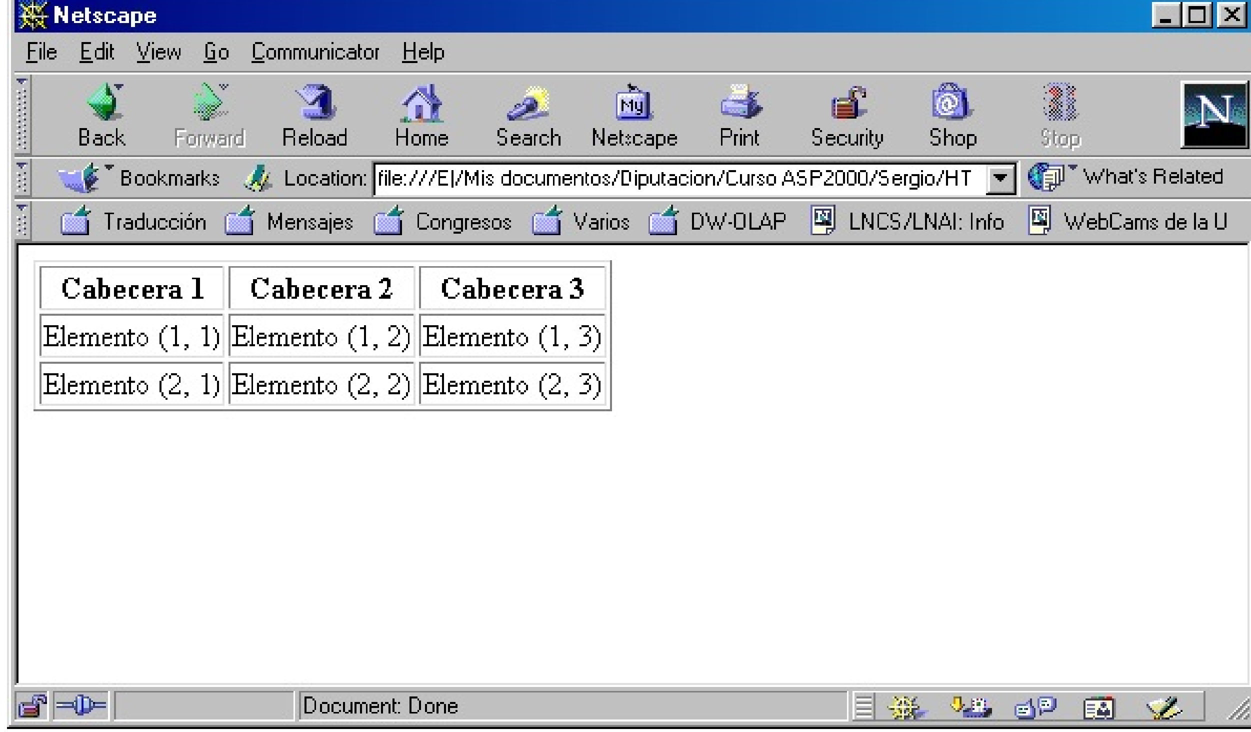


Figura 3.17: Tabla sencilla

Los atributos más importantes de la etiqueta <TABLE> son:

- BGCOLOR="color". Color de fondo de la tabla.
- BORDER="n". Indica el grosor del borde exterior de la tabla y de las líneas delimitadoras interiores. Un valor 0 produce que no tenga borde.
- CELLPADDING="n". Determina el espacio que debe existir entre los bordes de cada celdilla y el contenido de la celda.
- CELLSPACING="n". Especifica la cantidad de espacio que debe existir entre celdas contiguas.
- HEIGHT="n" | "n %". Alto de la tabla. Se puede indicar en pixels o en tanto por ciento en relación a la altura total disponible.
- WIDTH="n" | "n %". Ancho de la tabla. Se puede indicar en pixels o en tanto por ciento en relación a la anchura total disponible.

Los atributos más importantes de las etiquetas <TR>, <TH> y <TD> son:

- ALIGN="LEFT" | "CENTER" | "RIGHT". Alineamiento horizontal del contenido de una celda.
- vBGCOLOR="color". Color de fondo de una celda.
- VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP". Alineamiento vertical del contenido de una celda.

3.12.1. Tablas invisibles

Se conoce como tablas invisibles a aquellas que no poseen borde (BORDER="0"). Las tablas invisibles son muy útiles para distribuir los distintos elementos en una página **HTML**.

Por ejemplo, mediante tablas invisibles se puede mostrar el texto con márgenes a la izquierda y a la derecha, mostrar texto a varias columnas, dividir una imagen en diferentes ficheros y que se muestre como si no estuviese dividida, etc.

3.12.2. Tablas como marcos

Las tablas son muy útiles para crear marcos alrededor del texto o cualquier otro elemento de una página **HTML**. Se pueden conseguir efectos muy elegantes y a su vez sencillos de realizar mediante diversas tablas anidadas. Por ejemplo, el siguiente código, cuyo resultado se muestra en la Figura 3.18, muestra un texto sobre un fondo rojo rodeado de un marco amarillo en una página cuyo fondo es de color naranja. El tamaño de las celdas de la tabla exterior se ha modificado mediante el atributo CELLPADDING="10" para obtener un marco más ancho.

Ejemplo 3.22

```
<HTML>
<BODY BGCOLOR="orange">
<CENTER>
<TABLE BORDER="0" BGCOLOR="yellow" CELLPADDING="10">
<TR><TD>
	<TABLE BORDER="0" BGCOLOR="red">
	<TR><TD>
	<FONT SIZE="5">HTML útil y práctico</FONT>
	</TD></TR>
```

```
        </TABLE>
</TD></TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

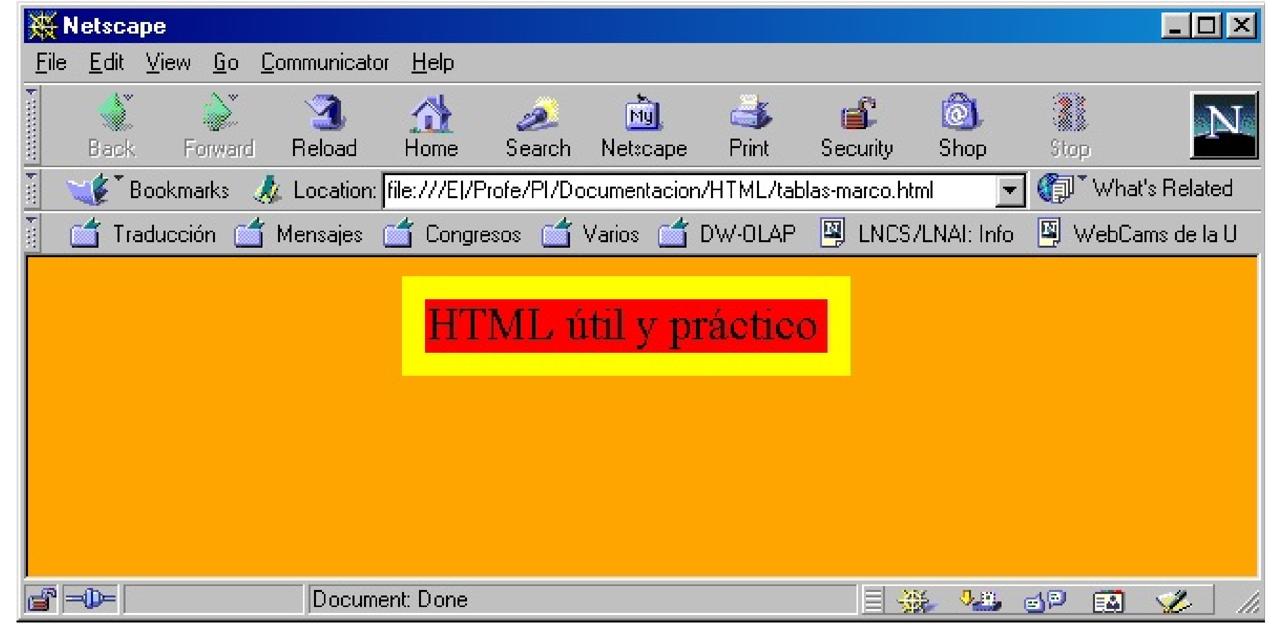


Figura 3.18: Tablas como marcos

3.13. Imágenes

Al inicio de la web, el empleo de imágenes en los documentos **HTML** era muy escaso‹ 23 ›. Sin embargo, hoy es todo lo contrario y es muy difícil encontrar páginas que no empleen una gran cantidad de imágenes.

Los dos formatos de imagen que admiten la mayoría de los navegadores son *Graphics Interchange Format* (**GIF**) y *Joint Photographic Experts Group* (**JPEG**). Las características básicas de ambos formatos se han resumido en el Cuadro 3.4.

Característica	GIF	JPEG
Colores	256 (8 bits)	16 777 216 (24 bits)
Transparencia	Sí	No
Animación	Sí	No
Compresión	Sin perdidas	Con perdidas
Dibujo	Sí	No
Fotografía	No	Sí

Cuadro 3.4: Diferencias entre GIF y JPEG

›.

También existe el formato **PNG**, que se ha creado como sustituto de **GIF**. Este formato gráfico se ha creado específicamente para Internet y otras redes de ordenadores. Sus características más importantes son: transparencia alfa, color de 24 bits y una compresión mejor que **GIF**. Además, emplea compresión sin pérdidas‹ 24 ›. Aunque Microsoft Internet Explorer y Netscape Communicator en sus últimas versiones ya lo soportan, aún no se encuentra muy extendido su uso.

3.13.1. Etiqueta <IMG>

La etiqueta **HTML** que permite insertar una imagen en un documento es <IMG>. Una imagen se puede colocar en cualquier punto de un documento: en un enlace, en una tabla, etc. Los atributos más importantes de esta etiqueta son:

- SRC="localización". Indica la localización y el nombre de la imagen que se quiere insertar.
- BORDER="anchura". Anchura del borde que se crea alrededor de la imagen. Cuando se coloca una imagen en un enlace, automáticamente se crea un borde; si no se quiere que aparezca el borde, hay que asignar el valor 0 a este atributo.
- WIDTH="anchura". Anchura de la imagen.
- HEIGHT="altura". Altura de la imagen.
- ALT="texto". Texto alternativo que se muestra si el navegador no admite la etiqueta <IMG> o se ha interrumpido la carga de imágenes.
- ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM". Especifica el alineamiento de la imagen con respecto al texto que la rodea.

Se recomienda indicar siempre la anchura y la altura de cada imagen con los atributos WIDTH y HEIGHT, ya que así la visualización de las páginas es más rápida‹ 25 ›.

El siguiente código muestra como se emplean las imágenes. La página está formada por una tabla con cuatro celdas. En cada celda se muestra la misma imagen con un alineamiento distinto. Además, la última imagen también posee un borde. En la Figura 3.19 se puede observar como se visualiza este código en un navegador.

Ejemplo 3.22

Ejemplo 3.23

```
<HTML>
<BODY>
<TABLE BORDER="0">
<TR>
	<TD>
	<IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="TEXTTOP">
	El manitas de la casa
	</TD>
	<TD>
	<IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="MIDDLE">
	El manitas de la casa
	</TD>
</TR>
<TR>
	<TD>
	<IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="BOTTOM">
	El manitas de la casa
	</TD>
	<TD>
	<IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" BORDER="10">
	El manitas de la casa
	</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

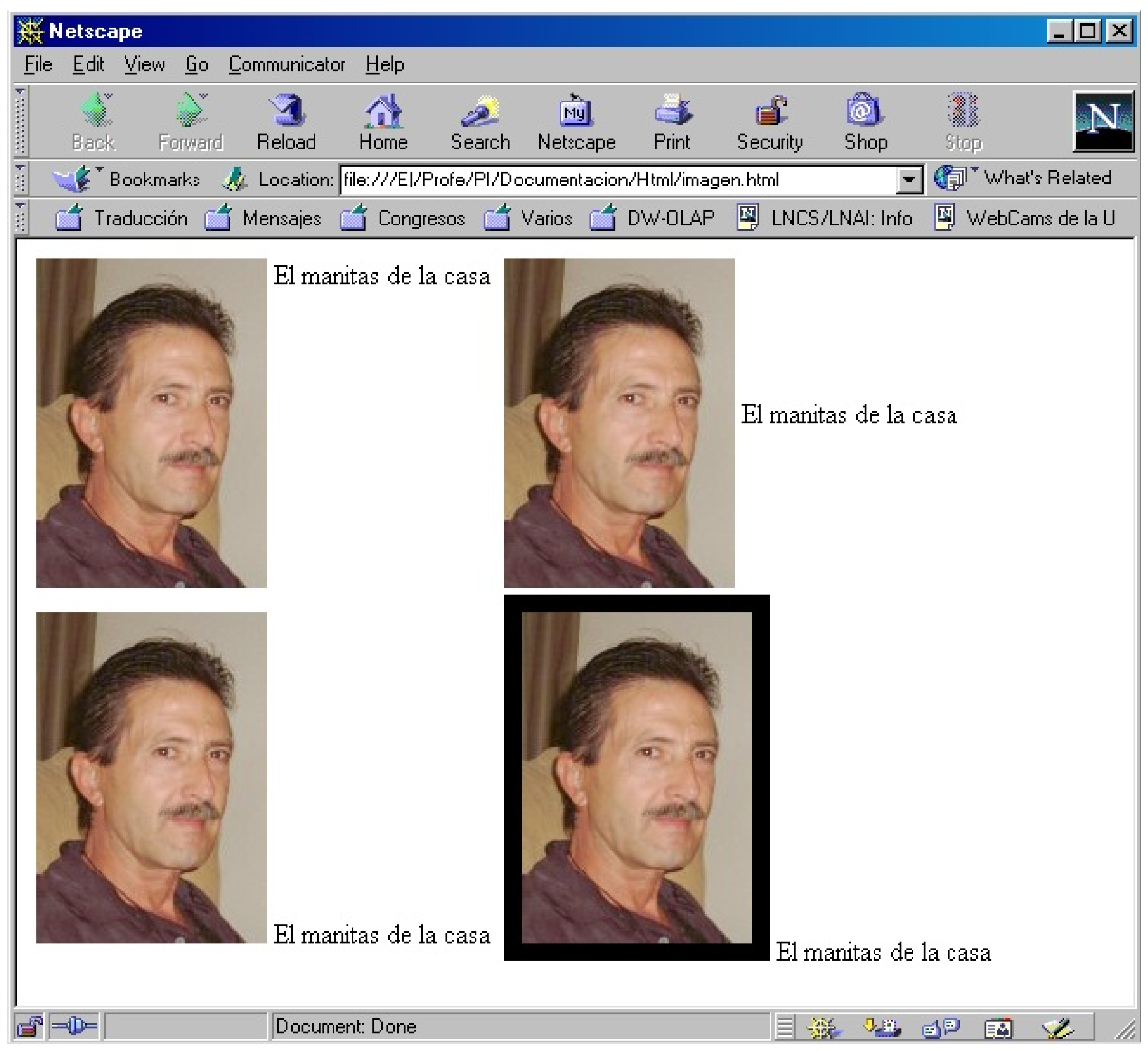


Figura 3.19: Imágenes con distinto alineamiento del texto

3.13.2. Imágenes como fondo de una página

La etiqueta <BODY> ... </BODY> posee el atributo BACKGROUND que permite mostrar una imagen como fondo de una página. Si la imagen tiene un tamaño menor que la ventana del navegador, la imagen se muestra en forma de "mosaico" hasta cubrir toda la superficie de la ventana.

3.14. Formularios

Los formularios son la herramienta que ofrece **HTML** para poder obtener

información de un usuario que visita una página **HTML** y enviarla al servidor web para su procesamiento.

Un formulario contiene dos tipos de elementos básicos: campos de datos (cuadros de texto, listas de selección, casillas de verificación) y de control (botones).

Dentro de una página **HTML** se puede incluir más de un formulario, pero teniendo en cuenta que no pueden anidarse ni solaparse. El servidor web sólo podrá recibir la información introducida en uno de ellos (sólo se envía la información de uno de los formularios al servidor).

La estructura básica de un formulario es:

Ejemplo 3.24

```
1 <FORM NAME="nombre" ACTION="pagina.html" METHOD="metodo">
2 Controles del formulario
3 </FORM>
```

El sentido de cada una de las líneas es:

- Línea 1: La etiqueta <FORM> marca el inicio del formulario. El atributo NAME asigna un nombre al formulario (para poder hacer referencia a él posteriormente), ACTION indica la dirección (**URL**) de la página o programa que procesa los datos del formulario en el servidor y METHOD indica el método que se va a utilizar para enviar los datos del formulario al servidor (GET o POST‹ 26 ›).
- Línea 2: En esta sección se incluyen los controles que posee el formulario.
- Línea 3: Fin del formulario.

3.14.1. Controles de un formulario

Un formulario puede contener los siguiente controles:

- Botones (para enviar información, borrar y otras acciones): <INPUT TYPE="SUBMIT">, <INPUT TYPE="RESET">, <INPUT TYPE="BUTTON">.
- Imágenes que actúan como botones (para enviar información): <INPUT TYPE="IMAGE">.
- Campos de verificación: <INPUT TYPE="CHECKBOX">.
- Campos excluyentes (botones de radio): <INPUT TYPE="RADIO">.

- Campos de texto: <INPUT TYPE="TEXT">.
- Campos de contraseña‹ 27 › (*password*): <INPUT TYPE="PASSWORD">.
- Campos ocultos: <INPUT TYPE="HIDDEN">.
- Envío de ficheros: <INPUT TYPE="FILE">.
- Listas de selección: <SELECT> ... </SELECT>, <OPTION>.
- Áreas de texto (campos de texto multilínea): <TEXTAREA> ... </TEXTAREA>.

Para que los datos introducidos en un formulario se envíen al servidor, todo formulario tiene que tener un botón de tipo TYPE="SUBMIT", que envía automáticamente los datos. Este botón se puede sustituir por un botón normal TYPE="BUTTON", pero entonces el envío se tiene que realizar manualmente mediante código de *script*.

Conviene dar un nombre a los campos que coloquemos en un formulario, ya que al enviar la información, ésta se transmite como pares nombre-valor. Para ello, todas las etiquetas de los controles poseen el atributo NAME para asignar un nombre al control, que deberá ser un nombre único, es decir, ningún otro control tendrá que tener el mismo nombre (excepto en el caso de los botones de radio), y el atributo VALUE para asignar un valor (todas las etiquetas tienen este atributo excepto <INPUT TYPE="IMAGE">, <SELECT> ... </SELECT> y <TEXTAREA> ... </TEXTAREA>). En los botones, el atributo VALUE modifica el texto que muestra el botón.

El siguiente código genera una página **HTML** con un formulario que contiene un campo de texto, un campo de contraseña, dos campos excluyentes con el mismo nombre (y por tanto sólo se puede elegir una de las dos opciones), dos campos de verificación, una lista de selección, un área de texto y dos botones (para enviar información y borrar). En la Figura 3.20 se muestra el formulario tal como se ve en un navegador.

Ejemplo 3.25

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML con formularios</TITLE>
</HEAD>
<BODY>
Esto es el cuerpo de una página HTML. Esta página posee un formulario:
<HR>
<FORM NAME="miFormulario" ACTION="procesa.asp" METHOD="POST">
Cuadro de texto: <INPUT TYPE="TEXT" NAME="control1a" VALUE="Algo">
<BR>
Cuadro de texto contraseña: <INPUT TYPE="PASSWORD" NAME="control1b">
```

```
<BR><BR>
Botones de radio:
<INPUT TYPE="RADIO" NAME="control2" VALUE="1">Opción 1
<INPUT TYPE="RADIO" NAME="control2" VALUE="2">Opción 2
<BR><BR>
Casilla de verificación:
<INPUT TYPE="CHECKBOX" NAME="control3a" VALUE="1">Opción 1
<INPUT TYPE="CHECKBOX" NAME="control3b" VALUE="2">Opción 2
<BR><BR>
Lista de selección:
<SELECT NAME="control4">
        <OPTION VALUE="ali">Alicante</OPTION>
        <OPTION VALUE="val">Valencia</OPTION>
        <OPTION VALUE="cas">Castellón</OPTION>
</SELECT>
<BR><BR>
Area de texto: <TEXTAREA NAME="control5"></TEXTAREA>
<BR><BR>
<INPUT TYPE="SUBMIT" VALUE="Enviar">
<INPUT TYPE="RESET" VALUE="Borrar">
</FORM>
<HR>
</BODY>
</HTML>
```

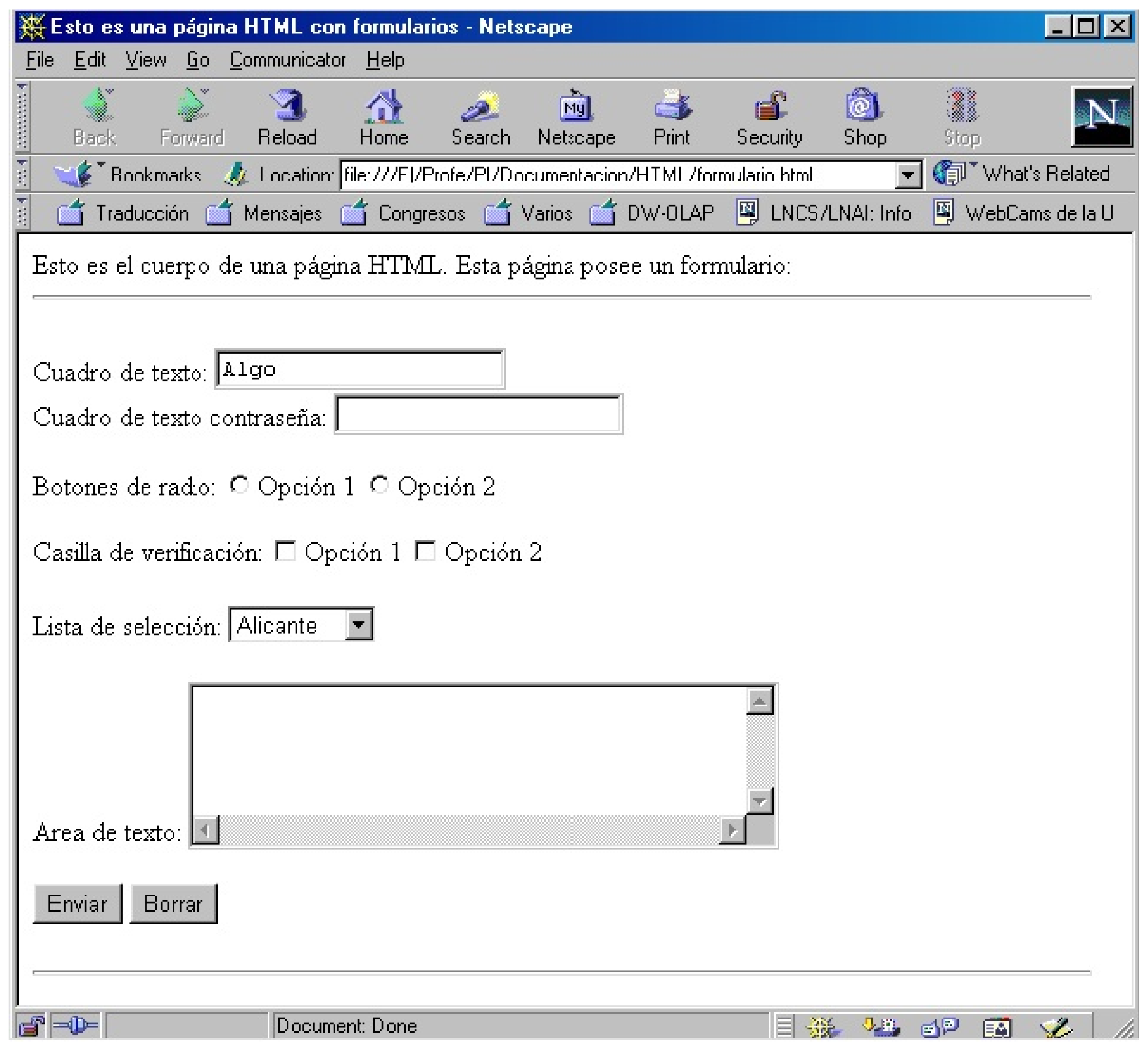


Figura 3.20: Formulario con distintos controles

3.14.2. Campos de verificación

Los campos de verificación (<INPUT TYPE="CHECKBOX">) poseen dos valores: activado y desactivado. Si al enviarse un formulario un campo de verificación está activo, se envía al servidor el valor indicado por VALUE. Distintos campos de verificación pueden tener el mismo nombre (NAME), aunque no es lo usual ya que "complica" la programación de la aplicación web en el servidor.

Si se desea que por defecto un campo de verificación aparezca activado, se tiene que incluir el atributo CHECKED en la etiqueta <INPUT TYPE="CHECKBOX">.

3.14.3. Campos excluyentes

Los campos excluyentes o botones de radio (<INPUT TYPE="RADIO">) tienen sentido cuando se emplean varios a la vez. Un grupo de botones de radio está formado por varios botones de radio que tienen todos el mismo nombre (NAME). En un grupo de botones de radio sólo un botón de radio puede estar seleccionado en un instante. Un formulario puede contener distintos grupos de botones de radio.

Los botones de radio también poseen el atributo CHECKED que permite indicar un botón de radio por defecto.

3.14.4. Campos de texto

En los campos de texto normal (<INPUT TYPE="TEXT">) y de contraseña (<INPUT TYPE="PASSWORD">) se puede escribir una cadena de caracteres. Se puede emplear el atributo SIZE para especificar el tamaño "visual" del cuadro de texto. Es decir, se puede indicar cuantos caracteres se pueden visualizar en un momento dado.

No confundir este atributo con MAXLENGTH, que especifica el número máximo de caracteres que se pueden introducir. Si no se especifica nada, se pueden introducir tantos caracteres como se desee.

3.14.5. Listas de selección

Las listas de selección se crean con la etiqueta <SELECT> ... </SELECT>. En las listas de selección se muestra una serie de opciones de las que el usuario puede elegir una. Si se añade el atributo MULTIPLE, el usuario puede elegir múltiples opciones‹ 28 ›. El atributo SIZE permite indicar cuantas opciones de la lista se visualizan simultáneamente.

Cada opción de una lista de selección se indica con la etiqueta <OPTION>. Cada opción puede tener asociado un valor (VALUE), que es el valor que se enviará al servidor. Si se desea que una opción aparezca marcada por defecto se tiene que añadir a la opción el atributo SELECTED.

El siguiente ejemplo muestra tres listas: una lista normal, una lista normal que muestra tres opciones a la vez y posee una seleccionada por defecto y una lista

múltiple. En la Figura 3.21 se puede observar como se visualiza este código en un navegador.

Ejemplo 3.26

```
<HTML>
<BODY>
<FORM>
Lista de selección normal:
<SELECT NAME="provincia">
	<OPTION VALUE="1">Alicante
	<OPTION VALUE="2">Valencia
	<OPTION VALUE="3">Castellón
</SELECT>
<BR><BR><BR>
Lista de selección normal de tamaño 3:
<SELECT NAME="universidad" SIZE="3">
	<OPTION VALUE="uv">Universidad de Valencia
	<OPTION VALUE="uji">Universidad Jaime I
	<OPTION VALUE="ua" SELECTED>Universidad de Alicante
	<OPTION VALUE="upv">Universidad Politécnica de Valencia
	<OPTION VALUE="umh">Universidad Miguel Hernández
	</SELECT>
<BR><BR><BR>
Lista de selección múltiple:
<SELECT NAME="departamento" MULTIPLE>
	<OPTION VALUE="dlsi">D. de Lenguajes y Sistemas Informáticos
	<OPTION VALUE="damma">D. de Análisis M. y M. Aplicada
	<OPTION VALUE="dfists">D. de Física, Ingeniería de Sistemas y ...
	<OPTION VALUE="dagr">D. de Análisis Geográfico Regional
	<OPTION VALUE="mmlab">Laboratorio Multimedia
</SELECT>
</FORM>
</BODY>
</HTML>
```

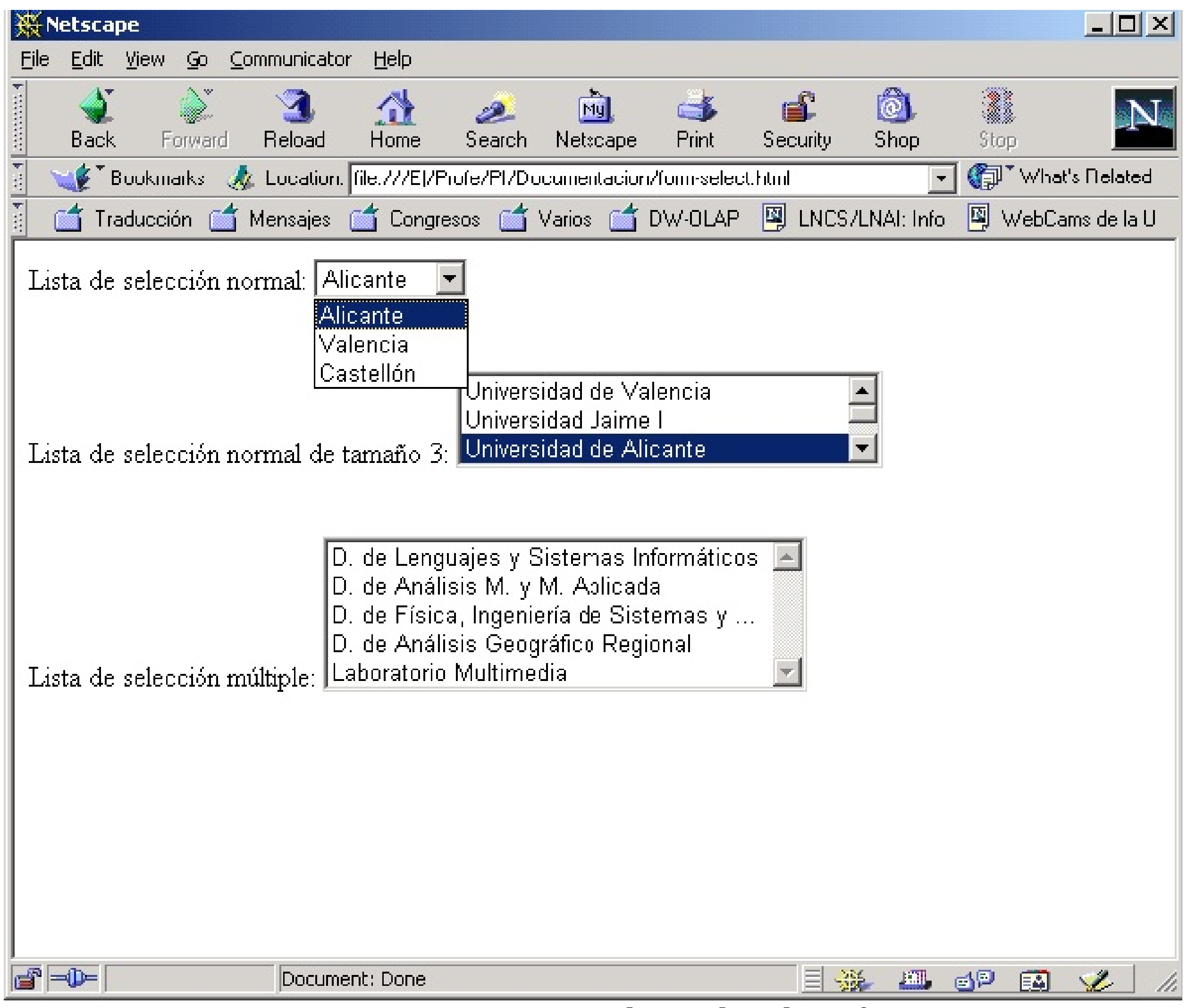


Figura 3.21: Distintas listas de selección

3.14.6. Áreas de texto

La etiqueta <TEXTAREA> ... </TEXTAREA> define un área de texto donde se pueden escribir varias líneas de texto. Esta etiqueta posee dos atributos que permiten modificar su tamaño. El atributo COLS indica el número de caracteres por línea que se pueden mostrar sin tener que realizar *scroll*. El atributo ROWS define el número de líneas que se pueden mostrar sin realizar *scroll*.

Si se quiere que el área de texto muestre un texto por defecto, se puede incluir entre las etiquetas de inicio y fin. El siguiente ejemplo muestra dos áreas de texto de distinto tamaño, una de ellas con un texto por defecto. En la Figura 3.22 se puede ver esta página visualizada en un navegador. Se pueden observar las barras de desplazamiento vertical y horizontal.

Ejemplo 3.27

```
<HTML>
<BODY>
<FORM>
Área 1:
<TEXTAREA ROWS="2" COLS="40">Texto por defecto...</TEXTAREA>
<BR>
Área 2:
<TEXTAREA ROWS="4" COLS="20"></TEXTAREA>
</FORM>
</BODY>
</HTML>
```

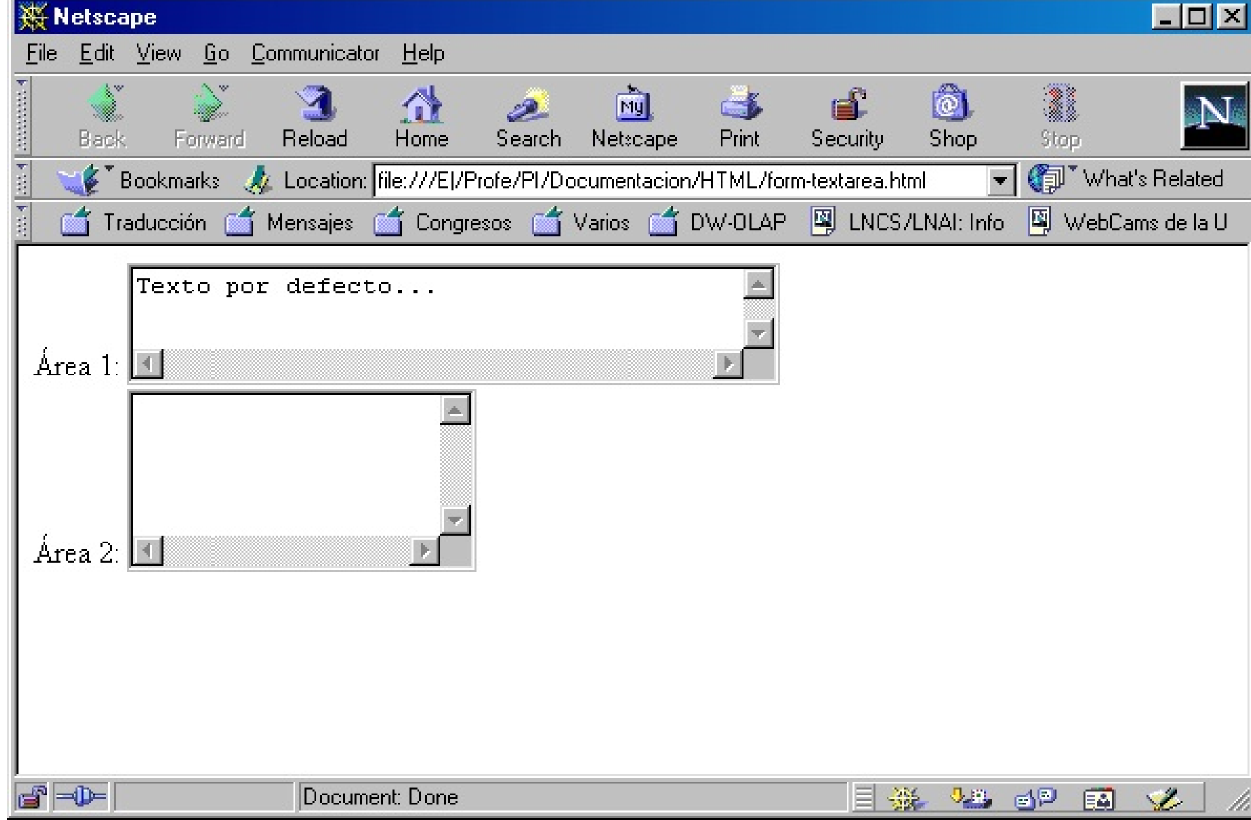


Figura 3.22: Áreas de texto de distinto tamaño

3.15. Marcos

Un marco (*frame*) es una región de una ventana que actúa como si fuera una ventana ella misma. La ventana principal puede contener múltiples marcos, de forma que diferentes regiones de la ventana muestren diferentes contenidos. A su vez, un marco puede contener otros marcos.

Normalmente, los marcos se emplean para dividir la ventana del navegador en dos partes: en una de ellas, la más pequeña, se muestra un índice del sitio web (esta

parte no varía); la otra, la más grande, modifica su contenido según los enlaces que se pulsen en el índice.

Para definir los marcos se emplean las tres etiquetas siguientes: <FRAMESET> ... </FRAMESET>, <FRAME> y <NOFRAMES> ... </NOFRAMES>.

La etiqueta <FRAMESET> ... </FRAMESET> define un conjunto de marcos que van a aparecer en la ventana. Esta etiqueta contiene una o más etiquetas <FRAME> que definen cada uno de los marcos. Los principales atributos de esta etiqueta son:

- BORDER="anchura". Anchura del borde de cada marco.
- FRAMEBORDER="YES" | "NO". Indica si el borde es en tres dimensiones o plano.
- COLS="listaAnchurasColumnas". Anchuras de cada uno de los marcos, separadas por comas. La anchura se puede indicar como número de pixels o como un porcentaje sobre el total disponible (en este caso, el número se tiene que acompañar del signo porcentaje). Por ejemplo, si se quiere dividir la ventana del navegador en dos columnas que ocupan el 30 y 70 por ciento:

Ejemplo 3.28

```
<FRAMESET COLS="30%,70%">
```

- ROWS="listaAlturasFilas". Alturas de cada uno de los marcos, separadas por comas. La altura se puede indicar como número de pixels o como un porcentaje sobre el total disponible (en este caso, el número se tiene que acompañar del signo porcentaje). Por ejemplo, si se quiere dividir la ventana del navegador en tres filas que ocupan el 10, 80 y 10 por ciento:

Ejemplo 3.29

```
<FRAMESET ROWS="10%,80%,10%">
```

La etiqueta <FRAMESET> ... </FRAMESET> define el número de filas (ROWS) o columnas (COLS) en que se va a dividir la ventana. Los dos atributos no se pueden emplear simultáneamente. Esta etiqueta se puede anidar, de forma que dentro de un <FRAMESET> se puede incluir otro <FRAMESET>. De este modo, se pueden combinar filas con columnas.

La etiqueta <FRAME> define un marco, que es una región de una ventana con contenido propio y que se puede navegar de forma independiente. Cada marco tiene su propia **URL** que define el contenido que se va a mostrar. Los atributos más importantes de esta etiqueta son:

- SRC="URL". Indica la **URL** del documento que se quiere mostrar en el marco.
- NAME="nombreMarco". Nombre del marco. Este nombre se emplea en la etiqueta

<A> ... </A> para mostrar una página en un marco concreto.

- FRAMEBORDER="YES" | "NO". Igual que el atributo de la etiqueta <FRAMESET> ... </FRAMESET>.
- NORESIZE. Si aparece este atributo, no se puede modificar el tamaño del marco.
- SCROLLING="YES" | "NO" | "AUTO". Indica si pueden aparecer barras de desplazamiento cuando no quepa toda la página en el marco.

Por último, la etiqueta <NOFRAMES> ... </NOFRAMES> se emplea para mostrar contenido en los navegadores que no pueden mostrar marcos. Los navegadores que sí que pueden mostrar marcos ignoran todo el contenido de esta etiqueta.

Lo interesante de los marcos es que se puede variar el contenido de los mismos de forma independiente. Desde un marco se puede modificar el contenido de otro. Para ello, se tiene que emplear un enlace (<A> ... </A>) con el atributo TARGET. Este atributo debe de tomar el nombre del marco que se quiere modificar (el nombre se habrá indicado en la etiqueta <FRAME> con el atributo NAME).

En el siguiente ejemplo, la página frame1.html crea una venta con dos marcos (marcoIzq y marcoDer). En el marco izquierdo se muestra la página frame1a.html; en el marco derecho se muestran las páginas frame1b.html y frame1c.html según se seleccione en los enlaces que contiene marcoIzq. En la Figura 3.23 vemos el resultado de mostrar estas páginas en un navegador.

- Página frame1.html:

Ejemplo 3.30

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<FRAMESET COLS="20%,80%" BORDER=10>
	<FRAME SRC="frame1a.html" NAME="marcoIzq">
	<FRAME SRC="frame1b.html" NAME="marcoDer">
	<NOFRAMES>
	Su navegador de Internet no permite mostrar marcos
	</NOFRAMES>
</FRAMESET>
</HTML>
```

- Página frame1a.html:

Ejemplo 3.31

```
<HTML>
<HEAD>
```

```
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<BODY>
Esta página tiene que aparecer a la izquierda.
<BR><BR>
Si pincha <A HREF="frame1b.html" TARGET="marcoDer">aquí</A>,
a la derecha tiene que aparecer la página 1.
<BR><BR>
Si pincha <A HREF="frame1c.html" TARGET="marcoDer">aquí</A>,
a la derecha tiene que aparecer la página 2.
</BODY>
</HTML>
```

- Página frame1b.html:

Ejemplo 3.32

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<BODY>
Esta es la página <B>1</B>.
</BODY>
</HTML>
```

- Página frame1c.html:

Ejemplo 3.33

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<BODY>
Esta es la página <B>2</B>.
</BODY>
</HTML>
```

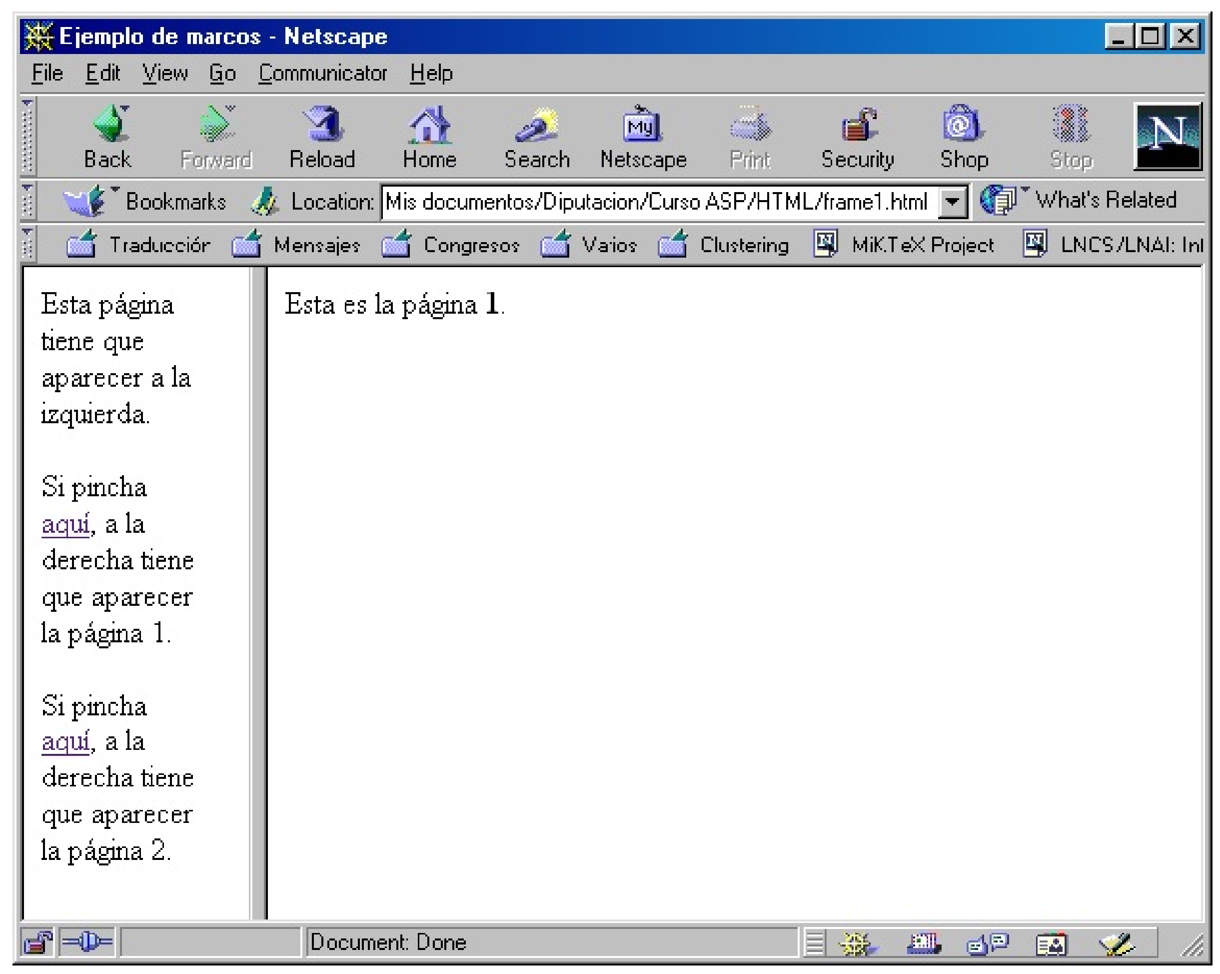


Figura 3.23: Página con dos marcos verticales

3.16. Guía de estilo

A continuación hemos incluido una serie de indicaciones que pueden ayudar a la hora de crear páginas web.

3.16.1. Organizar el código HTML

Si se crea una página web directamente con el código **HTML**, es recomendable hacerlo como si se estuviese programando. Es decir, organizar el código para que sea más fácil su lectura, poner comentarios, etc. Todo ello supone un trabajo extra, pero facilita el mantenimiento de las páginas.

3.16.2. Cuidado con los colores

Hay que llevar mucho cuidado con las combinaciones de colores que se emplean, ya que algunas cuestan mucho de leer.

Si tenemos páginas con mucho texto, es conveniente usar una combinación de colores de alto contraste, que facilite la lectura: un color oscuro sobre un fondo claro (negro o azul sobre blanco o un color crema) o al revés (un color claro para el texto sobre un fondo oscuro).

3.16.3. Cuidado con los tipos de letra

El uso de distintos tipos de letra puede originar bastantes problemas. Se suele recomendar un tipo de letra sans-serif (como Arial, Helvetica, Tahoma o Verdana) para la lectura de texto en pantalla.

A menudo, se elige un tipo de letra para un sitio web y aparece en algunas partes de los documentos texto en otro tipo de letra. No causa un buen efecto en el visitante "sufrir" este problema.

Por último, hay que intentar elegir tipos de letra "estándar". Si empleamos un tipo poco usual, seguramente la mayoría de la gente no lo tendrá instalado en sus ordenadores y las páginas no se visualizarán correctamente.

3.16.4. Sacar partido al hipertexto

Las referencias cruzadas permiten abordar un mismo tema en distintos niveles de profundidad. Se puede permitir al usuario pasar de largo información técnica o conceptos avanzados de un tema, estableciendo enlaces a textos más extensos. En general, podemos disponer de una página corta, con una presentación de la información escueta, que lleve a páginas que contengan detalles acerca de los temas tratados.

3.16.5. Usar las capacidades multimedia

Como se suele decir, una imagen vale más que mil palabras. Pero hay que tener cuidado: el uso de muchas imágenes puede confundir al usuario. Además, hay que tener en cuenta la velocidad de transferencia: las imágenes emplean muchos kilobytes, lo que aumenta el tiempo que tarda en cargarse una página. Hay que

ponerse como límite de 20 a 30 Kb de imágenes por página.

Un truco muy importante: una vez cargada una imagen, el navegador la almacena en la cache; cuando volvamos a emplear esa misma imagen en otra página, no necesitará descargarla otra vez. Para que nuestras páginas se carguen más rápidamente, es aconsejable combinar los mismos elementos gráficos (iconos, líneas, imágenes de fondo, etc.) en todas nuestras páginas.

Por último, mucho cuidado con las imágenes de fondo: si tienen muchos colores y detalles pueden ocultar otros detalles de la página e impedir la lectura correcta del texto.

3.16.6. Identidad corporativa

Hay que intentar conseguir una "identidad corporativa" en todas las páginas: emplear los mismos tipos de letra, colores, imágenes de fondo, iconos, etc., para dotar a las páginas de un estilo homogéneo. De esta forma, el usuario se sentirá "inmerso" en las páginas.

3.16.7. Permitir que los usuarios se comuniquen

En las páginas hay que incluir una dirección de contacto para que los usuarios se puedan comunicar. De este modo, se podrán recibir sugerencias, indicaciones de cómo mejorar las páginas o errores localizados.

3.16.8. Facilitar las búsquedas

Una adecuada clasificación y exposición de la información que contienen nuestras páginas facilitará la navegación de los usuarios. En general, la página principal (*homepage*) debería presentar en una sola página toda la información disponible en nuestro servidor.

3.16.9. Revisar las páginas periódicamente

No hay nada peor que encontrar enlaces rotos: enlaces que apuntan a páginas que

no existen, porque se hayan borrado, movido de sitio o falle la ruta de acceso. Una revisión periódica ahorra al usuario muchos problemas. Además, es importante indicar la fecha de la última modificación y las novedades añadidas. Esto permite, entre otras cosas, que el usuario se cerciore de la seriedad del autor de las páginas y facilita la navegación a los usuarios asiduos.

Si una página aparece con un signo "en construcción" y su última revisión es del año anterior, el usuario pensará que es mejor buscar lo que quiere en otra parte.

3.16.10. Los enlaces

La elección del lugar apropiado para poner los enlaces es crucial para una correcta presentación del hipertexto. Por ejemplo, compárense los dos siguientes trozos de código **HTML**. Es evidente que la primera opción es mucho mejor que la segunda.

Ejemplo 3.34

```
La <A HREF="/concejalias/turismo">Concejalía de Turismo</A> se
encarga de gestionar el turismo rural y de playa ...
```

Ejemplo 3.35

```
La Concejalía de Turismo se encarga de gestionar el turismo rural
y de playa ... (<A HREF="/concejalias/turismo">haga click aquí
para ver más información acerca de la Concejalía de Turismo</A>).
```

En los primeros años de la web había que poner haga click porque la gente no estaba acostumbrada a los enlaces, pero eso ya es historia.

Capítulo 4

Lenguajes de script

Los lenguajes de script permiten incluir "programación" en las páginas web. En este capítulo se explican las tres formas que existen de incluir y ejecutar código en una página web.

Índice General

4.1. Introducción

Un lenguaje de *script* (o lenguaje de guiones) es similar a un lenguaje de macros o a un fichero por lotes (*batch*): una lista de comandos que se pueden ejecutar sin o con la participación del usuario. Se trata en definitiva de un lenguaje de programación, que suele emplearse dentro de un contexto (dentro de una aplicación) y que no permite programar aplicaciones independientes (no permite crear ficheros ejecutables independientes, ya que los lenguajes de *script* normalmente son interpretados).

Existen multitud de lenguajes de *script* que se pueden emplear en páginas web: *JavaScript*, *VBScript*, *Perl*, *Rexx*, etc. Sin embargo, algunos sólo se pueden emplear en navegadores muy concretos y poco extendidos. Los lenguajes de *script* más empleados en Internet son *JavaScript* y en menor medida *VBScript*.

4.2. Diferencias entre VBScript y JavaScript

La diferencia más obvia entre *VBScript* y *JavaScript* (MICROSOFT lo denomina *JScript*) se encuentra en su sintaxis. *JavaScript* y *VBScript* nacieron con un mismo fin: dotar de un lenguaje de script rápido y sencillo a las páginas web. Cada uno de estos lenguajes

posee una serie de características que no posee el otro, pero ninguna de las diferencias existentes entre ambos permite descartar un lenguaje por el otro.

VBScript es un subconjunto de Visual Basic el lenguaje de programación "estrella" de MICROSOFT. Como *VBScript* se diseñó específicamente para trabajar con navegadores, no incluye características que se escapan del ámbito de los lenguajes de *script*, como el acceso a ficheros y la impresión.

JavaScript, por otro lado, deriva de la familia de lenguajes formada por *C*, *C++* y *Java*. Es conveniente aclarar desde un principio que *JavaScript* y *Java* son lenguajes totalmente distintos, desarrollados por distintas compañías (NETSCAPE y SUN MICROSYSTEMS respectivamente) y que, en principio, sólo comparten parte de la sintaxis y del nombre y poco más.

¿Qué lenguaje elegir? La principal razón para elegir uno u otro reside en la siguiente sencilla pregunta: **¿la plataforma que yo uso soporta ese lenguaje?** Mientras que los dos navegadores más extendidos, Microsoft Internet Explorer y Netscape Communicator, admiten *JavaScript*, sólo el primero admite *VBScript*. Por tanto, si elegimos este lenguaje de *script*, estaremos limitando el acceso a las páginas web que desarrollemos.

4.3. Para qué sirven

Las aplicaciones más habituales de los lenguajes de *script* en las páginas **HTML** son:

- Validar datos en el cliente y comprobar la consistencia de los valores antes de mandar un formulario (como, por ejemplo, comprobar que una fecha tiene un valor adecuado y un formato correcto).

- Actualizar campos relacionados en formularios (por ejemplo, establecer las opciones de una lista desplegable en función del valor seleccionado en unos botones de radio).

- Realizar procesamientos que no requieran la utilización de información centralizada (por ejemplo, convertir pesetas en euros, visualizar el calendario del mes actual, etc.).

- Servir de base para la utilización de otras tecnologías (**DHTML**, *Java*, *ActiveX*, etc.).

Los lenguajes de *script* también pueden actuar sobre el navegador a través de objetos integrados que representan al documento, la ventana activa, cada uno de los

controles de un formulario, etc. Para ello se emplea un modelo de objetos denominado *Document Object Model* (**DOM**), que veremos en el Capítulo 6.

Los lenguajes de *script* presentan fuertes restricciones de acceso a los recursos de la máquina en la que se ejecutan. Estas restricciones no se deben a limitaciones tecnológicas, sino a normas impuestas por los diseñadores de los lenguajes de *script* para evitar que la ejecución del código de una página web pueda dañar la integridad del sistema del usuario.

4.4. Como se usa un lenguaje de script en un navegador

Existen tres formas de incluir e invocar *scripts* dentro de una página **HTML**:

1. Usando la etiqueta <SCRIPT> ... </SCRIPT>. Esta etiqueta permite incluir código de *script* directamente en la página o que apunte a un fichero externo usando el atributo SRC. Esta etiqueta se puede incluir tanto en la sección HEAD como en BODY. Esta etiqueta posee el atributo LANGUAGE que permite indicar en que lenguaje (y que versión, si el lenguaje posee varias‹ 29 ›) se ha escrito el *script*. Si se omite este atributo, cada navegador tiene definido por defecto un lenguaje‹ 30 ›. El siguiente ejemplo muestra como combinar múltiples lenguajes de script en la misma página‹ 31 ›.

Ejemplo 4.1

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="VBScript">
        document.write "Esto lo ha escrito VBScript<BR>"
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
        document.write("Esto lo ha escrito JavaScript<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Mediante el atributo SRC se puede ocultar y proteger el código script de una página, aunque no es un sistema seguro, ya que cualquier usuario con unos conocimientos mínimos puede acceder al fichero que contiene el código. La ventaja principal de este sistema es que permite compartir el mismo código entre distintas páginas web. De este modo, se pueden crear librerías de código.

2. Usando los atributos de etiquetas **HTML** que soportan *scripts*. Existen una serie de elementos (<FORM>, <INPUT>, <SELECT>, <TEXTAREA>, <BODY> y <A>) pertenecientes al lenguaje **HTML** que permiten incluir código *script* en una serie de atributos que representan eventos. Cuando estos eventos se producen

(por ejemplo, al pulsar sobre un botón o al cargar una página), el código correspondiente se ejecuta. Estas etiquetas poseen el atributo LANGUAGE, que al igual que en la etiqueta SCRIPT, permite indicar en que lenguaje se ha escrito el código. El siguiente ejemplo muestra como gestionar los eventos de pulsación sobre botones (<INPUT TYPE="BUTTON">) mediante el atributo ONCLICK.

Ejemplo 4.2

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="VBScript">
        Sub PulsadoVBS
                alert "Pulsado el botón VBScript"
        End Sub
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
        function PulsadoJS()
        {
                alert("Pulsado el botón JavaScript");
        }
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="Formulario">
<INPUT TYPE="BUTTON" VALUE="VBScript" ONCLICK="PulsadoVBS"
LANGUAGE="VBScript">
<BR>
<INPUT TYPE="BUTTON" VALUE="JavaScript" ONCLICK="PulsadoJS()"
LANGUAGE="JavaScript">
</FORM>
</BODY>
</HTML>
```

3. Incluyendo código de *script* en una **URL**. Los *scripts* también se pueden invocar desde el elemento <A> ... </A> (*anchor*, ancla) mediante una **URL**. Esto permite que el *script* se ejecute cuando el usuario pulsa sobre un enlace. El siguiente ejemplo muestra una página con dos enlaces. Al pulsar sobre uno de ellos se abre una ventana nueva que muestra un mensaje.

Ejemplo 4.3

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="VBScript">
        Sub PulsadoVBS
                alert "Pulsado el enlace VBScript"
        End Sub
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
        function PulsadoJS()
```

```
        {
                alert("Pulsado el enlace JavaScript");
        }
</SCRIPT>
</HEAD>
<BODY>
<A HREF="javascript:PulsadoJS()">JavaScript</A>
<BR>
<A HREF="javascript:PulsadoVBS()">VBScript</A>
</BODY>
</HTML>
```

Como podemos observar en la **URL** del enlace, con la palabra javascript: indicamos el lenguaje que estamos empleando, y a continuación figura el código a ejecutar. Como vemos en el ejemplo, desde *JavaScript* se puede llamar a funciones escritas en *VBScript* y viceversa, pero como se ha comentado previamente, no se recomienda mezclar distintos lenguajes de *script* en una misma página.

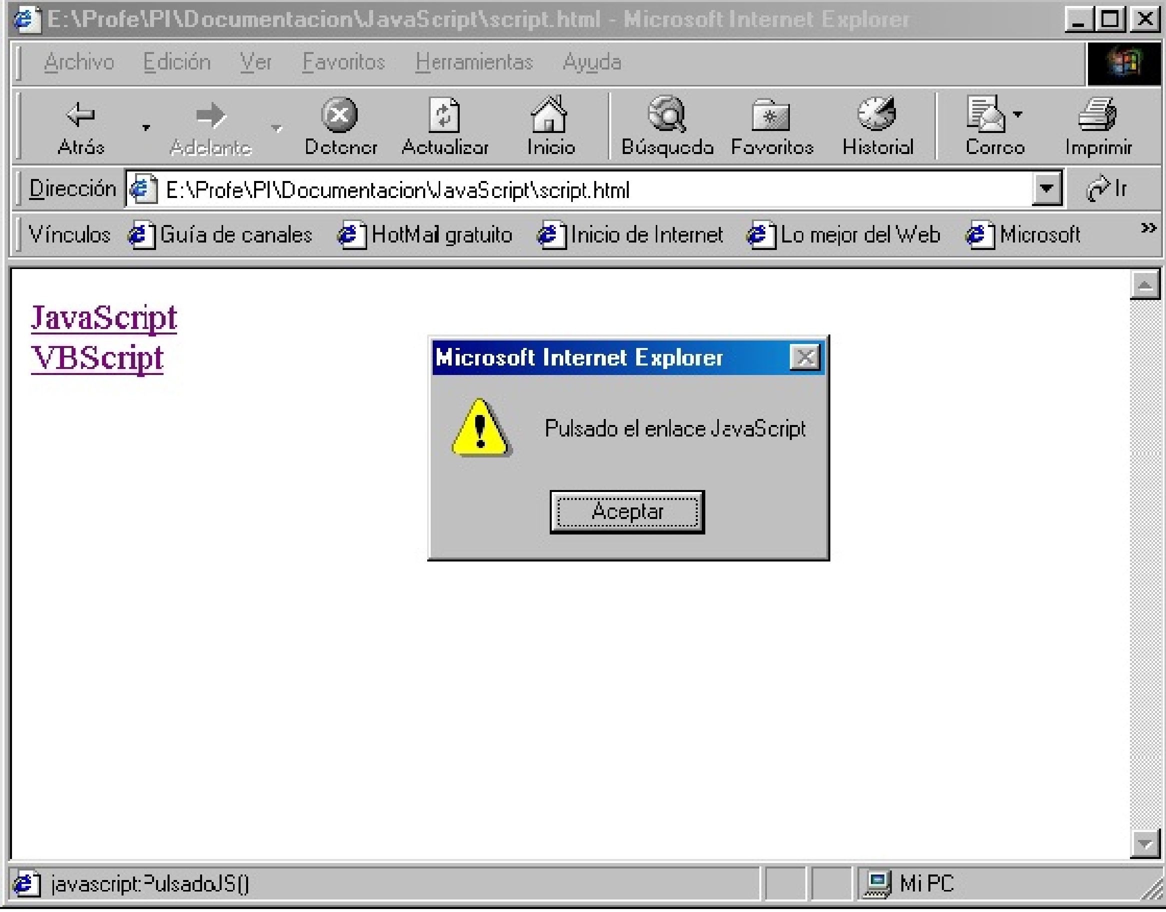


Figura 4.1: Código JavaScript en un enlace

Capítulo 5

JavaScript

En el capítulo anterior hemos visto los lenguajes de script en general. En este capítulo vamos a estudiar un lenguaje concreto: *JavaScript*. Este lenguaje es el más empleado en Internet y se puede considerar el lenguaje estándar. Veremos sus características básicas, sus diferentes sentencias, las funciones que incorpora y, por último, como validar formularios.

Índice General

5.1. Introducción

JavaScript es un lenguaje interpretado, basado en objetos (no es un lenguaje orientado a objetos "puro") y multiplataforma, inventado por NETSCAPE COMMUNICATIONS CORPORATION. Los navegadores de NETSCAPE fueron los primeros que usaron *JavaScript*. El primer nombre oficial de este lenguaje fue *LiveScript* y apareció por primera vez en la versión beta de Netscape Navigator 2.0 en septiembre de 1995, pero poco después fue rebautizado *JavaScript* en un comunicado conjunto con SUN MICROSYSTEMS el 4 de diciembre de 1995‹ 32 ›.

El núcleo de *JavaScript* (*Core JavaScript*) contiene una serie de objetos, como Array, Date, Math, Number y String, y un conjunto de elementos del lenguaje como operadores, estructuras de control y sentencias (%, ++, if, for, break, etc.). El núcleo de *JavaScript* se puede ampliar añadiendo objetos adicionales, como por ejemplo:

- *JavaScript* en el cliente (*Client-side JavaScript*) amplia el lenguaje añadiendo objetos que permiten controlar un navegador y su **DOM**.
- *JavaScript* en el servidor (*Server-side JavaScript*) amplia el lenguaje añadiendo objetos que son útiles cuando *JavaScript* se ejecuta en un servidor (lectura de ficheros, acceso a bases de datos, etc.).

JavaScript permite crear aplicaciones que se ejecuten a través de Internet, basadas en el paradigma cliente/servidor. La parte del cliente se ejecuta en un navegador, como Netscape Communicator o Microsoft Internet Explorer, mientras que la parte del servidor se ejecuta en un servidor, como Netscape Enterprise Server.

Mediante la característica *JavaScript's LiveConnect*, se pueden intercomunicar el código *JavaScript* con *Java*. Desde *JavaScript*, se pueden instanciar objetos *Java* y acceder a sus propiedades y métodos públicos. A su vez, desde *Java* se puede acceder a los objetos de *JavaScript*, a sus propiedades y a sus métodos.

5.1.1. Aplicaciones

Una de las aplicaciones principales de *JavaScript* consiste en validar la entrada introducida por el usuario a través de un formulario, que luego recibirán aplicaciones que se ejecutan en el servidor (hechas en **ASP**, **CGI**, **JSP** o cualquier otra tecnología). La utilidad de esto reside en:

- Reduce la carga en el servidor. Los datos incorrectos se filtran en el cliente y no se envían al servidor.
- Reduce los retrasos producidos por errores cometidos por el usuario. De otro modo la validación se tendría que realizar en el servidor, y los datos deberían viajar del cliente al servidor, ser procesados y entonces devueltos al cliente para que los corrigiese.
- Simplifica los programas que se ejecutan en el servidor al dividir el trabajo entre el cliente y el servidor.

Normalmente, la validación de los datos se puede realizar como mínimo en tres ocasiones:

- Cuando el usuario introduce los datos, con un manejador del evento onChange en cada control que se quiere validar del formulario.
- Cuando el usuario envía (*submit*) el formulario, con un manejador del evento onClick en el botón que envía el formulario.
- Cuando el usuario envía (*submit*) el formulario, con un manejador del evento onSubmit en el formulario.

Otra de las aplicaciones de *JavaScript* consiste en proporcionar dinamismo a las páginas **HTML**. Si se emplea junto con **DHTML** se pueden conseguir efectos sorprendentes.

5.1.2. Qué necesito para programar en JavaScript

Para poder programar en *JavaScript* no hace falta ningún equipo especial ni comprar un entorno de desarrollo. Sólo es necesario un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux y un navegador como Netscape Communicator o Microsoft Internet Explorer que soporte *JavaScript* para poder

comprobar el código.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Lo que sí es recomendable es utilizar un buen editor de textos, que sea cómodo, configurable, soporte macros, etc. y que sea *syntax highlight*. Esta última característica significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean variables, palabras reservadas, comentarios, etc.

5.1.3. JavaScript y Java

JavaScript y *Java* se confunden a veces entre sí. Son dos lenguajes similares en algunos aspectos, pero diferentes entre sí. *JavaScript* no tiene la estricta comprobación de tipos que posee *Java*, pero ambos si que comparten la sintaxis de las expresiones, de los operadores y de las estructuras de control.

JavaScript es un lenguaje interpretado, mientras que *Java* posee una fase de compilación (aunque no se genera código máquina, sino los llamados *bytecodes* que luego son interpretados por la *máquina virtual Java*). *JavaScript* tiene un pequeño número de tipos de datos: números (enteros y en coma flotante), booleanos y cadenas. *JavaScript* posee un modelo de objetos basado en prototipos mientras que el de *Java* se basa en clases.

JavaScript es un lenguaje con mucha libertad si se compara con *Java*. No es necesario declarar las variables, clases y métodos. No hay que preocuparse con métodos públicos, privados o protegidos, y no hay que implementar interfaces. En el Cuadro 5.1 se resumen las diferencias existentes entre *JavaScript* y *Java*.

JavaScript	**Java**
Interpretado (no compilado) por el cliente	Compilado a *bytecodes* que se descargan desde el servidor y se ejecutan en el cliente.
Basado en objetos. No distingue entre tipos de objetos. La herencia se realiza a través del mecanismo de prototipado, y las propiedades y los métodos se pueden añadir a cualquier objeto de forma dinámica.	Basado en clases. Distinción entre clases e instancias, la herencia se realiza a través de la jerarquía de clases. No se pueden añadir a las clases ni a las instancias propiedades o métodos de forma dinámica.
Código integrado y embebido con el código **HTML**.	Los *applets* se distinguen del código **HTML** (aunque se acceden a través

código **HTML**.	de **HTML**).
El tipo de dato de las variables no se declara (tipos dinámicos).	El tipo de dato de las variables se tiene que declarar (tipos estáticos).

Cuadro 5.1: JavaScript frente a Java

5.1.4. Versiones

Cada versión de los navegadores soporta una versión diferente de *JavaScript*. Cada nueva versión de *JavaScript* añade nuevas características al lenguaje (y corrige fallos de las anteriores). El Cuadro 5.2 muestra las distintas versiones de *JavaScript* soportadas por las diferentes versiones de Netscape Navigator. Las versiones anteriores a la 2.0 no soportaban *JavaScript*. Actualmente se encuentra en desarrollo *JavaScript 2.0*.

Versión de JavaScript	**Versión de Navigator**
JavaScript 1.0	Navigator 2.0
JavaScript 1.1	Navigator 3.0
JavaScript 1.2	Navigator 4.0 - 4.05
JavaScript 1.3	Navigator 4.06 - 4.78
JavaScript 1.4	Ningún navegador (sólo en los productos de servidor de Netscape)
JavaScript 1.5	Navigator 6.0 y Mozilla

Cuadro 5.2: Relación entre las versiones de JavaScript y de Netscape Navigator

Si queremos detectar que versión de JavaScript admite nuestro navegador podemos emplear el siguiente código:

Ejemplo 5.1

```
<HTML>
<HEAD>
<TITLE>Ejemplo version JavaScript</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
<!--
```

```
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
<!--
        version = "JavaScript 1.1"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
        version = "JavaScript 1.2"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.3">
<!--
        version = "JavaScript 1.3"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.4">
<!--
        version = "JavaScript 1.4"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.5">
<!--
        version = "JavaScript 1.5"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!--
        document.write("Soporta " + version);
// -->
</SCRIPT>
</BODY>
</HTML>
```

Este código funciona de la siguiente forma: mediante el atributo LANGUAGE de la etiqueta <SCRIPT> ... </SCRIPT> se indica la versión de *JavaScript* que se emplea. Como las versiones aparecen de menor a mayor, cuando se llegue a una versión que no se admita, la variable version almacenará la última versión admitida, que será la mayor de todas las que se admiten.

En el ejemplo anterior se puede ver que las instrucciones de *JavaScript* aparecen entre comentarios de **HTML** (<!== Comentario ==>). Este sistema impide que el código de *script* se visualice en aquellos navegadores que no entienden la etiqueta <SCRIPT> ... </SCRIPT>. En el resto de ejemplos de este libro no se incluye, pero es una "buena práctica" hacerlo siempre.

5.1.5. JavaScript y ECMA

Aunque NETSCAPE inventó *JavaScript* y sus navegadores fueron los primeros que lo soportaron, NETSCAPE trabaja estrechamente con *European Computer Manufacturers Association* (**ECMA**) para obtener un lenguaje de programación estándar basado en *JavaScript*.

La versión estandarizada de *JavaScript*, llamada *ECMAScript* se "debe de" comportar de forma idéntica en todas las aplicaciones que soportan el estándar. La primera versión de *ECMAScript* se detalla en la especificación *ECMA-262 Edition 1*. Este estándar también ha sido aprobado por *International Organization for Standards* (**ISO**) bajo la denominación *ISO-16262*.

Versión de JavaScript	**Versión de ECMA**
JavaScript 1.1	ECMA-262 Edition 1 se basa en JavaScript 1.1
JavaScript 1.2	ECMA-262 Edition 1 no se había completado cuando apareció JavaScript 1.2 (la compatibilidad entre ambos no es total).
JavaScript 1.3	JavaScript 1.3 es completamente compatible con ECMA-262 Edition 1, pero posee características adicionales que no forman parte de ECMA-262 Edition 1.
JavaScript 1.4	JavaScript 1.4 es completamente compatible con ECMA-262 Edition 1, pero con características adicionales. Cuando apareció, aún no se hab ía completado ECMA-262 Edition 3.
JavaScript 1.5	JavaScript 1.5 es completamente compatible con ECMA-262 Edition 3.

Cuadro 5.3: Relación entre las versiones de JavaScript y de ECMA

JavaScript siempre incluirá características que no formen parte de la especificación de **ECMA**. Pero estas características no impiden que *JavaScript* sea compatible con **ECMA**.

5.1.6. JScript

JScript es otro lenguaje de *script* que puede emplearse tanto en el cliente (dentro de páginas **HTML**) como en el servidor (en páginas **ASP**). *JScript* es la implementación

páginas **HTML**) como en el servidor (en páginas **ASP**). *JScript* es la implementación realizada por Microsoft del lenguaje de script estándar *ECMA-262*.

Un aspecto importante a tener en cuenta es que la versión del lenguaje de *script* es independiente de la versión del navegador. Por ejemplo, se puede instalar el motor de *JScript v5* en Microsoft Internet Explorer 3 y usar todas las características del lenguaje (por seguridad, no se puede hacer lo contrario, instalar *JScript v1* en Microsoft Internet Explorer 5).

Versión de JScript	**Productos de Microsoft**
JScript v1	Internet Explorer 3
JScript v2	Internet Information Server 3.0
JScript v3	Internet Explorer 4
JScript v4	Visual Interdev 6.0
JScript v5	Windows 2000, Office 2000 e Internet Explorer 5

Cuadro 5.4: Relación entre las versiones de JScript y los productos de Microsoft

5.1.7. Diferencias entre JavaScript y JScript

En las primeras versiones, *JavaScript* y *JScript* poseían muchas diferencias entre ellos. Sin embargo, en las últimas versiones las diferencias prácticamente han desaparecido, ya que ambos aseguran que cumplen el estándar *ECMA-262*. Pero aún quedan pequeñas diferencias que pueden hacer "perder el tiempo" al programador. Por ejemplo, el siguiente código produce distintos resultados al ejecutarse en Netscape Communicator o Microsoft Internet Explorer. La diferencia se encuentra en que el primero convierte automáticamente a entero el valor de la variable a, que almacena una cadena, mientras que el segundo navegador no lo hace.

Ejemplo 5.2

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
a = "1";

switch(a)
{
	case 0:
		document.write("0");
```

```
        case 1:
            document.write("1");
            break;

        case 2:
            document.write("2");
            break;

        default:
            document.write("Ninguno");
            break;
}
</SCRIPT>
</BODY>
</HTML>
```

La salida que produce Netscape Communicator 4.7 es: 1.

La salida que produce Microsoft Internet Explorer 5.0 es: Ninguno.

5.2. El lenguaje

5.2.1. Características básicas

La sintaxis de *JavaScript* es prácticamente idéntica a la de *C*, *C++* y *Java*‹ 33 ›. Si se conoce alguno de estos lenguajes, aprender *JavaScript* resulta muy sencillo y requiere poco esfuerzo. Los comentarios, las funciones, las sentencias de control, etc., poseen la misma sintaxis que en *C*. *JavaScript* es un lenguaje basado en objetos, pero no orientado a objetos como *Java* y *C++*. El objetivo de *JavaScript* es conseguir código sencillo y reducido, no crear programas grandes y complejos.

JavaScript es un lenguaje *case sensitive* (sensible a minúsculas/mayúsculas). Esto quiere decir que todas las palabras que usemos para referirnos a los distintos identificadores del lenguaje tendrán que ser escritas correctamente para que sean comprendidas por el intérprete de *JavaScript*. Por ejemplo, en el siguiente código se declaran tres variables que son distintas:

Ejemplo 5.3

```
var nombre, Nombre, NOMBRE;
```

En *JavaScript* no es necesario acabar las sentencias con un punto y coma (sí que es obligatorio en *C*), excepto para separar código que se encuentra en la misma línea. Pero es una buena recomendación acabar siempre las sentencias con punto y coma. Por ejemplo, las tres líneas siguientes de código son correctas:

Ejemplo 5.4

```
var nombre; var apellido1
var apellido2
var ciudad;
```

Al igual que en *C*, el código se estructura por bloques (*block-structured computer language*). Un bloque de código se indica delimitando las sentencias que lo componen mediante las llaves ({ y }). Por ejemplo, las siguientes sentencias forman un bloque de código:

Ejemplo 5.5

```
{
        document.write("Una sentencia ");
        document.write("en un ");
        document.write("bloque de código");
}
```

Los espacios en blanco, tabuladores y saltos de línea no poseen valor en *JavaScript*, por lo que el código anterior se podría escribir de otra forma, tal como se ve en el siguiente ejemplo, y sería totalmente equivalente. El "aspecto visual" del código depende del estilo que cada uno quiera emplear.

Ejemplo 5.6

```
{
        document.write("Una sentencia "); document.write("en un ");
                        document.write("bloque de código");}
```

5.2.2. Comentarios

Los comentarios son totalmente transparentes al código, y no afectan al mismo ni en el modo de ejecutarse ni en la velocidad de ejecución. Los comentarios se implementan de dos maneras diferentes.

La primera es utilizando dos barras inclinadas (//). Desde que el intérprete de *JavaScript* encuentra las dos barras hasta que llega al final de la línea, todo el texto que haya entremedias será ignorado.

La segunda manera de comentar fragmentos de código es usando los símbolos barra inclinada y asterisco (/*) y viceversa (*/). Estos símbolos definen zonas de comentario que pueden extenderse a través de varias líneas.

Ejemplo 5.7

```
// Esta linea esta comentada
var a; // El comentario no afecta al codigo

/* Todo este codigo esta comentado
var b = 0;
alert("Hola a todos");
*/
```

5.2.3. Declaración de variables

Para definir variables en *JavaScript* usaremos la palabra reservada var. Al contrario que en otros lenguajes, no es necesario declarar las variables que se emplean, pero es una buena costumbre.

No todas las palabras sirven para definir variables. Existen tres normas que hay que cumplir:

- Un nombre de variable válido se compone únicamente de una palabra (sin espacios), pudiendo estar formada por letras, números o el guión de subrayado (_).
- La primera letra del nombre de la variable deberá ser siempre una letra o el carácter de guión de subrayado. No podrá ser un número.
- El nombre de la variable no debe coincidir con ninguna de las palabras reservadas.

Al declarar una variable es posible asignarle un valor inicial. Si no se le asigna un valor inicial, posee un valor nulo representado por la palabra clave null. El empleo de variables sin valor puede producir errores.

Ejemplo 5.8

```
// Declaraciones correctas
var _nombre, _apellido1, _apellido2;
var Provincia = "Alicante";
var ciudad1, ciudad2;
var codigo_postal = "03001";

// Declaraciones incorrectas
```

```
var 12edad;
var var;
```

JavaScript es un lenguaje sin tipos, es decir, no necesita una declaración del tipo de las variables para trabajar con ellas, porque él mismo se encarga de reconocer que es lo que se quiere almacenar. Los distintos tipos de datos que puede almacenar una variable en *JavaScript* son:

- **Cadenas**. Las cadenas de texto se delimitan por comillas dobles (") o comillas simples (') y pueden contener cualquier tipo de combinación de números, letras, espacios y símbolos. El uso de los dos tipos de comillas tiene su sentido, ya que debido a que *JavaScript* se usa conjuntamente con el lenguaje **HTML**, y como éste usa las comillas dobles en su sintaxis, tiene que haber algún método de distinguir las cadenas de *JavaScript* de las cadenas de **HTML**. Por ejemplo: var ciudad = "Elche", provincia = "Alicante";.

- **Números enteros**. Un número entero se representa como cualquier sucesión de dígitos (0 a 9), precedidos por el signo menos (-) si es un número negativo. Por ejemplo: var a = 1, b = 100000, c = -123456789;.

- **Números en coma flotante**. Se representan de igual forma que los números enteros, pero añaden una parte decimal que se representa con un punto (.) seguido de tantas cifras como compongan la parte decimal del número. Por ejemplo: var a = 9.12, b = -3.141592, c = 10000000000000.1;.

- **Booleanos**. Representa únicamente dos valores, verdadero (true) o falso (false) y se suele utilizar en sentencias condicionales. Por ejemplo: var rojo = true, verde = false;.

- **Nulo**. Se utiliza únicamente para comprobar si una variable que se ha definido tiene un valor asignado o no. Por ejemplo:

Ejemplo 5.9

```
var nombre;
if(nombre == null)
{
        alert("Introduzca su nombre, por favor");
}
```

5.2.4. Ámbito de las variables

Ya se ha comentado antes que no es necesario declarar las variables antes de

usarlas. Sin embargo, el hecho de declarar una variable sí que influye en el ámbito de existencia (dónde existe y cuánto tiempo existe una variable). Siempre que se declare una variable local a un ámbito (a una función, por ejemplo), dominará sobre cualquier otra variable que se hubiese declarado fuera de ese ámbito. Además, una variable declarada en una función no puede ser accedida desde otra función.

Por ejemplo, en el siguiente código se declaran tres variables globales a, b y c. Aunque la variable b no haya sido declarada con var, el solo hecho de asignarle un valor ya supone declararla. En la función fun(), se declara una variable c local ("Maria"), por lo que no se puede acceder a la variable c global ("Ana"). Desde esta función se llama a la función fun2(); desde esta función sí que se puede acceder a la variable global c.

Ejemplo 5.10

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var a = "Juan";
b = "Jose";
var c = "Ana";

function fun2()
{
	document.write("fun2-c- " + c + "<BR>");
}

function fun()
{
	var c = "Maria";

	document.write("fun1-a- " + a + "<BR>");
	document.write("fun1-b- " + b + "<BR>");
	fun2();
	document.write("fun1-c- " + c + "<BR>");
}

fun();

document.write("-c-" + c + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

El código anterior produce la siguiente salida en un navegador:

Ejemplo 5.11

```
fun1-a- Juan
fun1-b- Jose
fun2-c- Ana
```

```
fun1-c- Maria
-c- Ana
```

5.2.5. Caracteres especiales

Además de los caracteres normales, en una cadena también se pueden incluir caracteres especiales que permiten generar saltos de líneas o caracteres a partir de su código **ASCII**. El Cuadro 5.5 muestra los caracteres especiales que se pueden emplear dentro de una cadena.

Carácter	Significado
\b	Retroceso (backspace)
\f	Salto de página (form feed)
\n	Salto de línea (new line)
\r	Retorno de carro (carriage return)
\t	Tabulador
\'	Apóstrofe o comilla simple
\"	Comilla doble
\\	Barra invertida (backslash)
\XXX	El carácter de la codificación Latin-1 especificado por los tres dígitos octales entre 0 y 377.
\xXX	El carácter de la codificación Latin-1 especificado por los dos dígitos hexadecimales entre 00 y FF.
\uXXXX	El carácter Unicode especificado por los cuatro dígitos hexadecimales entre 0000 y FFFF.

Cuadro 5.5: Caracteres especiales

Por ejemplo, el siguiente código produce el resultado que se muestra en la Figura 5.1.

Ejemplo 5.12

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write("\\251 produce: \251<BR>");
document.write("\\xAA produce: \xAA<BR>");
document.write("\\u00FA produce: \u00FA<BR>");
</SCRIPT>
</BODY>
```

```
</HTML>
```

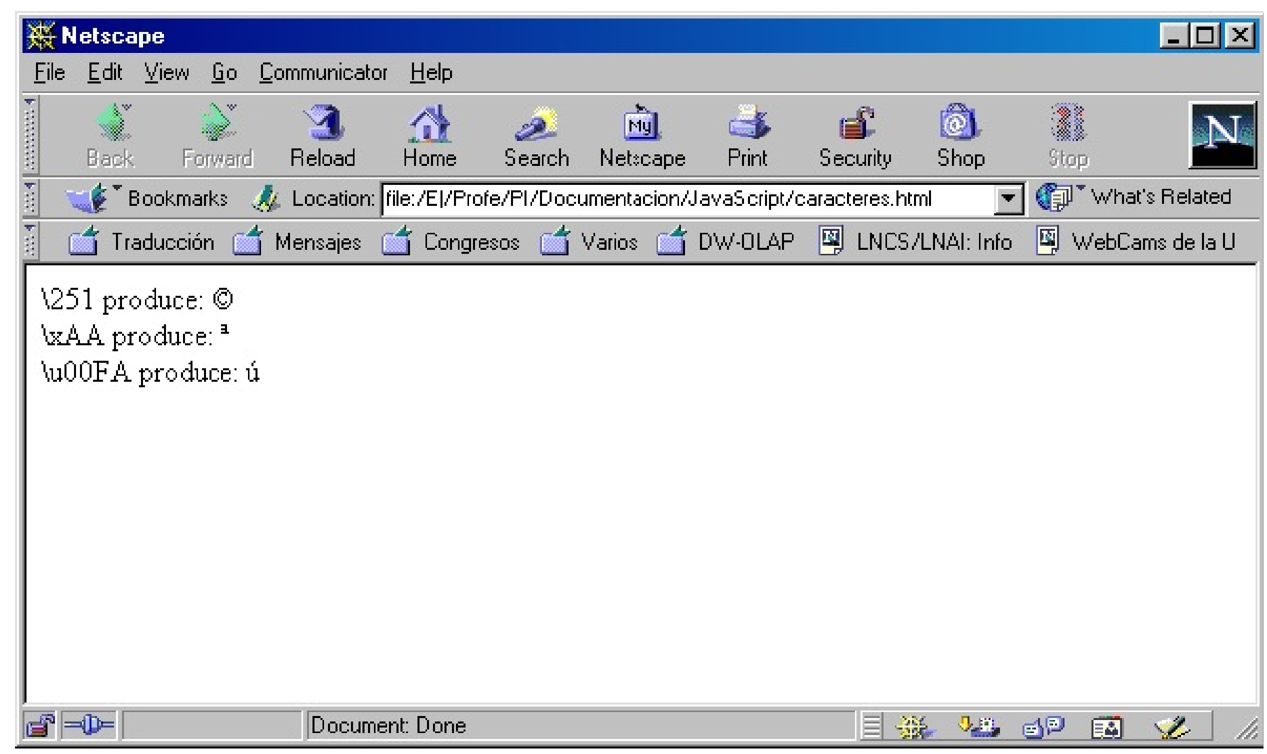


Figura 5.1: Ejemplo de caracteres especiales

5.2.6. Operadores

Los operadores son los signos que se usan para referirse a distintas operaciones que se pueden realizar con las variables y con las constantes, tales como suma, resta, incremento, etc.

La precedencia de los operadores determina el orden en que se aplican cuando se evalúa una expresión. Se puede anular la precedencia usando paréntesis. El Cuadro 5.6 muestra la precedencia de los operadores, de menos a más. Los operadores que se encuentran en un mismo nivel poseen la misma precedencia.

Tipo de operador	**Operadores**
Coma	,
Asignación	= += -= *= /= %= <<= >>= >>>= &= ^= |=
Condicional	?:
O lógico (OR)	||
Y lógico (AND)	&&
O bit a bit (OR)	|
O exclusiva bit a bit (XOR)	^
Y bit a bit (AND)	&

Igualdad	== != === !==
Relacional	< <= > >=
Desplazamiento bit a bit	<< >> >>>
Suma y resta	+ -
Multiplicación y división	* / %
Negación e incremento	! ~ - + ++ -- typeof void delete
Llamada a función	()
Creación de instancias	new
Miembro	. []

Cuadro 5.6: Precedencia de los operadores de JavaScript

5.2.7. Palabras reservadas

Las palabras reservadas (*reserved words*) no se pueden usar como nombres de variables, funciones, métodos u objetos. Algunas de las palabras reservadas son palabras clave (*keywords*) de span *JavaScript*, mientras que otras se han reservado para futuros usos.

En el Cuadro 5.7 se muestran las palabras reservadas de span *JavaScript 1.3*.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	else
enum	export	extends	false
final	finally	float	for
function	goto	if	implements
import	in	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	

Cuadro 5.7: Palabras reservadas de JavaScript

5.3. Sentencias

Vamos a detallar las sentencias que incluye el lenguaje *JavaScript*. Todas estas sentencias son anidables: se pueden incluir unas dentro de otras tantas veces como se quiera.

5.3.1. Condicionales

Una sentencia condicional es un conjunto de comandos que se ejecutan si una condición es cierta (true). *JavaScript* proporciona dos sentencias condicionales: if ... else y switch.

if . . . else

La sentencia de selección simple posee la siguiente sintaxis (la parte entre corchetes es opcional):

Ejemplo 5.13

```
if(condicion)
{
        sen1
}
[else
{
        sen2
}]
```

El conjunto de sentencias sen1 se ejecuta si condicion es cierta; en caso contrario se ejecuta sen2.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.14

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var a;

a = prompt("Escriba un número, por favor", "");
if(a <= 1)
{
```

```
	alert("Es menor o igual que 1");
}
else if(a <= 5)
{
	alert("Es mayor que 1, pero menor o igual que 5");
}
else
{
	alert("Es mayor que 1");
}
</SCRIPT>
<TITLE>Ejemplo if</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
</BODY>
</HTML>
```

switch

La sentencia de selección múltiple posee la siguiente sintaxis:

Ejemplo 5.15

```
switch(expresion)
{
	case et1 :
		sen1;
		[break;]
	case et2 :
		sen2;
		[break;]
	...
	[default :
		sen;]
}
```

La sentencia switch evalúa una expresión e intenta encontrar una etiqueta (valor) que case con el resultado de la evaluación. Si se logra casar, se ejecutan las sentencias asociadas con esa etiqueta. Si no se logra ningún emparejamiento, se busca la etiqueta opcional default y se ejecutan sus sentencias asociadas.

La sentencia break opcional en cada etiqueta asegura que el programa salga del switch una vez que se han ejecutado las sentencias correspondientes y continúa la ejecución por la línea siguiente a la sentencia switch. Si no se incluye un break, el flujo de ejecución pasará de una opción case a otra hasta que encuentre un break o finalice la sentencia switch.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.16

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var a;

a = prompt("¾Cuántos hijos tiene?", "");
switch(a)
{
	case 0: alert("Anímese un día de estos");
		break;
	case 1: alert("Puede tener más");
		break;
	case 2: alert("Ya tiene la parejita, ahora a por el trío");
		break;
	default: alert("Una familia como las de antes: muchos hijos");
		break;
}
</SCRIPT>
<TITLE>Ejemplo switch</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
</BODY>
</HTML>
```

5.3.2. De repetición

Un bucle es un conjunto de comandos que se ejecutan repetidamente hasta que una condición especificada deja de cumplirse. *JavaScript* ofrece tres tipos de sentencias de repetición: for, do ... while y while.

for

La sintaxis de la repetición con contador es la siguiente:

Ejemplo 5.17

```
for ([expInicial]; [condicion]; [expIncremento])
{
	sentencias
}
```

Cuando un bucle for se ejecuta, ocurre lo siguiente:

1. Si existe la expresión de inicialización expInicial, se ejecuta. La expresión inicial

suele inicializar uno o más contadores, pero se pueden emplear expresiones de cualquier grado de complejidad.

2. Se evalúa la expresión condicion.

3. Si el valor de condicion es cierto (true), las sentencias del bucle se ejecutan. Si el valor de condicion es falso (false), el bucle termina.

4. La expresión de actualización expIncremento se ejecuta y el flujo de ejecución vuelve al paso 2.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.18

```
<HTML>
<HEAD>
<TITLE>Ejemplo for</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
var a;

for(a = 0; a <= 10; a++)
{
        document.write("Vamos por el número " + a + "<BR>\n");
}
</SCRIPT>
</BODY>
</HTML>
```

do . . . while

La sentencia de repetición con condición final posee la siguiente sintaxis:

Ejemplo 5.19

```
do
{
        sentencias
} while(condicion)
```

Las sentencias se ejecutan siempre al menos una vez antes de que condicion se evalúe. Si se evalúa a cierto, se repite el proceso: las sentencias se ejecutan una vez más y condicion se vuelve a evaluar.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.20

```
<HTML>
<HEAD>
<TITLE>Ejemplo do while</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a, b;

a = 1;
do
{
	b = a;
	do
	{
		document.write(" ");
		b--;
	}
	while(b > 0);
	document.write(a + "<BR>\n");
	a++;
}
while(a < 10);
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

while

La sintaxis de la sentencia de repetición con condición inicial es la siguiente:

Ejemplo 5.21

```
while(condicion)
{
	sentencias
}
```

Si condicion es falsa, las sentencias del bucle dejan de ejecutarse y el control pasa a la siguiente sentencia después del bucle. La evaluación de condicion ocurre antes de que se ejecuten sentencias. Por tanto, puede ocurrir que las sentencias no se ejecuten nunca.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.22

```
<HTML>
<HEAD>
<TITLE>Ejemplo while</TITLE>
```

```
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a, b;

a = 1;
while(a <= 10)
{
	document.write("Tabla del " + a + "\n");
	document.write("===========\n");
	b = 1;
	while(b <= 10)
	{
		document.write(a + " x " + b + " = " + (a * b) + "\n");
		b++;
	}
	document.write("\n");
	a++;
}
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

break

La sintaxis de esta sentencia es:

Ejemplo 5.23

```
break
o
break etiqueta
```

La sentencia break se usa para finalizar un bucle (for, do..while, while) o una sentencia de selección múltiple (switch).

continue

La sintaxis de esta sentencia es:

Ejemplo 5.24

```
continue
o
continue etiqueta
```

La sentencia continue se usa para pasar a la siguiente iteración en un bucle (for, do ... while, while).

5.3.3. De manipulación de objetos

Existen dos sentencias de manipulación de objetos: for(... in ...) y with.

for(. . . in . . .)

La sintaxis de esta sentencia es:

Ejemplo 5.25

```
for(variable in objeto)
{
	sentencias
}
```

Esta sentencia permite que una variable recorra todas las propiedades de un objeto. Para cada propiedad del objeto, se ejecutan las sentencias. Para acceder a las propiedades de un objeto se emplea el operador punto (.).

Como ejemplo tenemos el siguiente código:

Ejemplo 5.26

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function Propiedades(obj, obj_nombre)
{
	var resultado = "";

	for (var i in obj)
	{
		resultado += obj_nombre + "." + i + " = " + obj[i] + "<BR>"
	}

	return resultado;
}
</SCRIPT>
<TITLE>Ejemplo for_in</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
document.write(Propiedades(window, "window"));
</SCRIPT>
```

```
</PRE>
</BODY>
</HTML>
```

with

La sintaxis de esta sentencia es:

Ejemplo 5.27

```
with(objeto)
{
        sentencias
}
```

La sentencia with establece el objeto por defecto sobre el que se ejecutan un conjunto de sentencias que pueden acceder a las propiedades o métodos del objeto. De este modo, se evita tener que escribir el nombre del objeto cada vez.

Como ejemplo tenemos el siguiente código:

Ejemplo 5.28

```
<HTML>
<HEAD>
<TITLE>Ejemplo with</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
with(Math)
{
        document.write("E: " + E + "\n");
        document.write("LN10: " + LN10 + "\n");
        document.write("LN2: " + LN2 + "\n");
        document.write("LOG10E: " + LOG10E + "\n");
        document.write("LOG2E: " + LOG2E + "\n");
        document.write("Pi: " + PI + "\n");
        document.write("SQRT1_2: " + SQRT1_2 + "\n");
        document.write("SQRT2: " + SQRT2 + "\n");
}
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

5.4. Funciones

Una función es un fragmento de código al que suministramos unos datos determinados (parámetros o argumentos), ejecuta un conjunto de sentencias y puede devolver un resultado.

5.4.1. Declaración de funciones

La definición de una función se compone de las siguientes partes:

- La palabra clave function.
- El nombre de la función.
- Una lista de argumentos, encerrados entre paréntesis y separados por comas. Una función puede tener cero o más argumentos.
- Las sentencias que definen la función, encerradas entre llaves. Las sentencias dentro de una función pueden llamar a su vez a otras funciones o a la misma función (recursividad).

Por tanto, la sintaxis de una función es:

Ejemplo 5.29

```
function nombre([arg1 [, arg2 [, ...]]])
{
        sentencias
}
```

En general, todas las funciones se deberían de definir en la sección <HEAD> ... </HEAD> de la página. Así, cuando el usuario carga la página, las funciones es lo primero que se carga. De otro modo, el usuario podría realizar alguna acción (por ejemplo, lanzar un evento) que llamase a una función que no ha sido definida aún, lo que produciría un error.

Por ejemplo, el siguiente código define dos funciones llamadas cuadrado y cubo que calculan el cuadrado y el cubo del número que reciben como argumento respectivamente:

Ejemplo 5.30

```
<HTML>
<HEAD>
<TITLE>Ejemplo function</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a;

function cuadrado(numero)
{
	return numero * numero;
}

function cubo(numero)
{
	return numero * numero * numero;
}

a = prompt("Escribe un número entero:", "");
alert("El cuadrado de " + a + " es " + cuadrado(a));
a = prompt("Escribe un número entero:", "");
alert("El cubo de " + a + " es " + cubo(a));
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

La instrucción return se emplea para finalizar la ejecución de una función y también para devolver un valor.

5.4.2. Funciones predefinidas

JavaScript posee un conjunto de funciones predefinidas: eval, isFinite, isNaN, parseInt, parseFloat, Number, String, escape y unescape.

eval

Evalúa una expresión que contiene código de *JavaScript*. La sintaxis es eval(expr), donde expr es la expresión que va a ser evaluada.

Si la cadena representa una expresión, se evalúa; si representa una o más sentencias, se ejecutan. No hace falta llama a eval para evaluar expresiones aritméticas, ya que *JavaScript* evalúa las expresiones aritméticas automáticamente. El siguiente ejemplo muestra el funcionamiento de esta función:

Ejemplo 5.31

```
<HTML>
<HEAD>
<TITLE>Ejemplo eval</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
var a = 1, b = 2, c = 3, d = 4, e = 5, v;

v = prompt("Escriba una letra de la 'a' a la 'e'", "");
alert("El valor de '" + v + "' es " + eval(v));
</SCRIPT>
</BODY>
</HTML>
```

isFinite

Determina si el argumento representa un número finito. La sintaxis es isFinite(num) donde num es el numero que se quiere evaluar.

Si el argumento es "NaN" (*not a number*, no un número), devuelve false, en caso contrario devuelve true. Por ejemplo, el siguiente código permite filtrar la entrada del usuario y averiguar si ha introducido un número correcto o cualquier otra cosa:

Ejemplo 5.32

```
<HTML>
<HEAD>
<TITLE>Ejemplo isFinite</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
v = prompt("Escriba un número", "");
if(isFinite(v))
	alert("Correcto, es un número");
else
	alert("Te has equivocado");
</SCRIPT>
</BODY>
</HTML>
```

isNaN

Determina si el argumento es "NaN" (*not a number*, no un número). La sintaxis es isNaN(num) donde num es el numero que se quiere evaluar.

Las funciones parseFloat y parseInt devuelven "NaN" cuando evaluan un valor que no es un número. La función isNaN devuelve true si el argumento es "NaN" y false en caso

contrario.

parseInt y parseFloat

Las dos funciones de conversión parseInt y parseFloat devuelven un valor numérico a partir de una cadena.

La sintaxis de parseFloat es parseFloat(cad), donde cad es la cadena que se quiere convertir a un número en coma flotante. Si encuentra un carácter devuelve el valor hallado hasta ese punto e ignora ese carácter y el resto. Si el primer carácter no puede convertirse a un número, devuelve "NaN" (*not a number*, no un número).

La sintaxis de parseInt es parseInt(cad [, base]), donde cad es la cadena que se quiere convertir a la base indicada en el segundo argumento opcional. Por ejemplo, 10 indica que se convierta a decimal, 8 a octal, 16 a hexadecimal. Para bases mayores que diez, las letras del alfabeto indican numerales mayores que nueve. Por ejemplo, para los números hexadecimales (base 16), las letras de la A a la F se usan. Si encuentra un carácter que no es un numeral válido en la base especificada, ese carácter y todos los caracteres que le siguen se ignoran y devuelve el valor convertido hasta ese punto. Si el primer carácter no puede convertirse a un número en la base especificada, devuelve "NaN" (*not a number*, no un número).

Number y String

Las funciones Number y String convierten un objeto a un número o a una cadena, respectivamente. La sintaxis de estas funciones es Number(objRef) y String(objRef), donde objRef es una referencia a un objeto. Por ejemplo, el siguiente código convierte un objeto Date a un número y a una cadena:

Ejemplo 5.33

```
<HTML>
<HEAD>
<TITLE>Ejemplo Number-String</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
// Obtiene la fecha actual
var d = new Date();

// Milisegundos transcurridos desde el 1 de enero de 1970
alert("Milisegundos transcurridos: " + Number(d));

// Formato mas adecuado
alert("Fecha: " + String(d));
</SCRIPT>
</BODY>
</HTML>
```

escape y unescape

Las funciones escape y unescape permiten codificar y decodificar cadenas. La función escape devuelve la codificación hexadecimal de su argumento codificado en **ISO** Latin-1. La función unescape devuelve la cadena correspondiente a la codificación hexadecimal pasada como argumento. La sintaxis de estas funciones es escape(string) y unescape(string). Por ejemplo, el siguiente código genera una página con un formulario con dos botones. Un botón aplica la función escape a la cadena que introduce el usuario y el otro aplica la función unescape.

Ejemplo 5.34

```
<HTML>
<HEAD>
<TITLE>Función predefinida escape()</TITLE>
<SCRIPT LANGUAGE="JavaScript"> function hazEscape()
{
        var cadena;

        cadena = prompt("Escriba una cadena", "");
        alert(escape(cadena));
}

function hazUnescape() {
        var cadena;

        cadena = prompt("Escriba una cadena", "");
        alert(unescape(cadena));
}
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<FORM NAME="formulario">
<INPUT TYPE="BUTTON" VALUE="escape" ONCLICK="hazEscape()">
<INPUT TYPE="BUTTON" VALUE="unescape" ONCLICK="hazUnescape()">
</FORM>
</CENTER>
</BODY>
</HTML>
```

5.5. Objetos

JavaScript es un lenguaje basado en objetos. Pero no se puede considerar que sea "orientado a objetos", ya que no posee características básicas como la herencia simple o múltiple, la jerarquía de clases, el polimorfismo, la ocultación de datos, etc.

En *JavaScript* un objeto es un constructor con *propiedades* (que pueden ser variables u otros objetos) y *métodos* (funciones asociadas a objetos).

En la Sección 5.3.3 se vieron las sentencias for(... in ...) y with que permiten manipular objetos. Para acceder a las propiedades o métodos de un objeto se emplea el operador punto (.). Por ejemplo, en el siguiente código se accede a la propiedad status y al método alert del objeto window:

Ejemplo 5.35

```
window.status = "Bienvenido a JavaScript";
window.alert("2 + 2 = " + (2 + 2));
```

También se puede acceder a las propiedades y métodos de un objeto mediante la notación de *arrays asociativos*. Se puede acceder al objeto como si fuera un array. Cada índice de acceso es una cadena que representa una propiedad o un método.

El código del ejemplo anterior se expresa mediante arrays asociativos de la siguiente forma:

Ejemplo 5.36

```
window.["status"] = "Bienvenido a JavaScript";
window.["alert"]("2 + 2 = " + (2 + 2));
```

5.5.1. Creación de objetos

Existen dos formas de crear un objeto nuevo: inicializadores de objetos y funciones constructoras.

Inicializadores de objetos

La sintaxis de este método es:

Ejemplo 5.37

```
nombreObjeto = {prop1:val1, prop2:val2, ..., propN:valN}
```

donde nombreObjeto es el nombre del nuevo objeto, propI es el nombre de una propiedad y valI el valor asociado a la propiedad. Un objeto puede tener como propiedad otro objeto.

Por ejemplo, el siguiente código define dos objetos llamados profesor y asignatura. El objeto profesor forma parte del objeto asignatura.

Ejemplo 5.38

```
profesor = {nombre:"Sergio", apellidos:"Luján Mora"}
asignatura = {cod:"PI", cred:6, prof:profesor};
document.write("Código: " + asignatura.cod);
document.write("<BR>");
document.write("Créditos: " + asignatura["cred"]);
document.write("<BR>");
document.write("Profesor: " + asignatura.prof.nombre
+ " " + asignatura.prof.apellidos);
document.write("<BR>");
```

Funciones constructoras

En este sistema se emplea una función que realiza el papel de "constructor". Cuando se crea el objeto, se emplea el operador new y el nombre de la función constructora. Este método es mejor que el anterior, ya que permite crear un "tipo" para distintos objetos.

En la función constructora se indican las propiedades y métodos del objeto. Una característica especial de *JavaScript* es que en cualquier momento se pueden añadir nuevas propiedades o métodos a un objeto concreto (no se añaden a todos los objetos del mismo tipo). Dentro de la función constructora se emplea la palabra reservada this para hacer referencia al objeto.

Por ejemplo, el siguiente código define un objeto de tipo Persona y dos objetos de tipo Pagina. Estos últimos poseen una propiedad que es un objeto de tipo Persona.

Ejemplo 5.39

```
function Persona(nombre, apellidos)
{
	this.nombre = nombre;
	this.apellidos = apellidos;
}

function Pagina(autor, url, fecha)
{
	this.autor = autor;
	this.url = url;
	this.fecha = fecha;
}

slm = new Persona("Sergio", "Luján Mora");
p1 = new Pagina(slm, "index.html", "18/04/2001");
p2 = new Pagina(slm, "home.html", "23/03/2001");
```

```
document.write(p1.url + ": " + p1.fecha);
document.write("<BR>");
document.write(p2.url + ": " + p2.fecha);
```

5.5.2. Métodos de un objeto

Para definir un método de un objeto, simplemente hay que asignar a una propiedad del objeto el nombre de una función. En la función, si se quiere acceder a las propiedades del objeto, se tiene que emplear la palabra reservada this.

Por ejemplo, el siguiente código crea dos objetos de tipo Ordenador. Estos objetos poseen un método llamado ver que muestra las propiedades del objeto.

Ejemplo 5.40

```
function ver()
{
	document.write("CPU: " + this.cpu + "<BR>");
	document.write("Velocidad: " + this.vel + " MHz<BR>");
	document.write("Memoria: " + this.mem + " Mb<BR>");
	document.write("Disco duro: " + this.dd + " Gb<BR>");
}

function Ordenador(cpu, vel, mem, dd)
{
	this.cpu = cpu;
	this.vel = vel;
	this.mem = mem;
	this.dd = dd;
	this.ver = ver;
}

o1 = new Ordenador("Pentium", 200, 32, 3);
o2 = new Ordenador("Pentium II", 500, 128, 15);
o1.ver();
o2.ver();
```

5.5.3. Eliminación de objetos

Para eliminar un objeto se emplea el operador delete. Este operador devuelve true si el borrado es correcto y false en caso contrario.

5.6. Tratamiento de cadenas

Las cadenas en *JavaScript* se pueden representar de dos formas:

- Directamente como literales: caracteres encerrados entre comillas simples o dobles.
- Como un objeto String: este objeto "envuelve" al tipo de dato cadena.

No se debe confundir una cadena literal con un objeto String. Por ejemplo, el siguiente código crea la cadena literal s1 y el objeto de tipo String s2:

Ejemplo 5.41

```
// crea una cadena literal
s1 = "algo"

// crea un objeto String
s2 = new String("algo")
```

Todos los métodos del objeto String se pueden emplear directamente en una cadena literal: *JavaScript* convierte automáticamente la cadena literal a un objeto String temporal, ejecuta el método sobre ese objeto y finaliza eliminando el objeto String temporal. También se puede emplear la propiedad String.length con una cadena literal: devuelve el número de caracteres (longitud) que posee la cadena.

A continuación mostramos los principales métodos que posee el objeto String: charAt, concat, indexOf, lastIndexOf, replace, slice y split.

charAt(indice)

Devuelve el carácter especificado por indice en la cadena. El indice toma valores entre 0 y la longitud de la cadena menos 1. Ejemplo:

Ejemplo 5.42

```
var cadena = "Hola";

document.writeln("Carácter en 0 es " + cadena.charAt(0) + "<BR>");
document.writeln("Carácter en 1 es " + cadena.charAt(1) + "<BR>");
document.writeln("Carácter en 2 es " + cadena.charAt(2) + "<BR>");
```

El ejemplo anterior produce la siguiente salida:

Ejemplo 5.43

```
Carácter en 0 es H
```

```
Carácter en 1 es o
Carácter en 2 es l
```

concat(cadena1, cadena2, . . . , cadenaN)

Concatena varias cadenas en una sola. Ejemplo:

Ejemplo 5.44

```
var cad1 = "Hola";
var cad2 = " a";
var cad3 = " todo";
var cad4 = " el mundo";

cad5 = cad1.concat(cad2, cad3, cad4);
document.writeln(cad5);
```

El código anterior produce el siguiente resultado:

Ejemplo 5.45

```
Hola a todo el mundo
```

indexOf(valorBuscado [, inicio])

Devuelve la primera posición de valorBuscado dentro de la cadena a partir del principio de la cadena (o de inicio que es opcional) o -1 si no se encuentra. Ejemplo:

Ejemplo 5.46

```
var cuenta = 0;
var pos;
var cad = "Cadena con unas letras"

pos = cad.indexOf("a");
while(pos != -1)
{
	cuenta++;
	pos = cad.indexOf("a", pos + 1);
}
document.write("La cadena contiene " + cuenta + " aes");
```

El ejemplo anterior produce la siguiente salida:

Ejemplo 5.47

```
La cadena contiene 4 aes
```

lastIndexOf(valorBuscado [, inicio])

Similar a la función indexOf, pero busca desde el final de la cadena o el inicio que se indique hacia el principio. Ejemplo:

Ejemplo 5.48

```
var pos;
var cad = "Cadena con unas letras"

pos = cad.lastIndexOf("a");
document.write("La última a está en " + pos + "<BR>");
pos = cad.lastIndexOf("a", pos - 1);
document.write("La penúltima a está en " + pos + "<BR>");
```

Este ejemplo genera el siguiente resultado:

Ejemplo 5.49

```
La última a está en 20
La penúltima a está en 13
```

replace(exRegular, nuevaCadena)

Busca una expresión regular (exRegular) en una cadena y cada vez que se encuentra se sustituye por nuevaCadena. Si se quiere buscar varias veces, hay que incluir el modificador g en la expresión regular para realizar una búsqueda global. Si no se quiere distinguir minúsculas y mayúsculas, hay que incluir el modificador i. Ejemplo:

Ejemplo 5.50

```
var cad = "Juan es hermano de Juan Luis";
var aux;

aux = cad.replace(/Juan/gi, "Jose");
document.write(cad + "<BR>");
document.write(aux + "<BR>");
```

El código anterior muestra el siguiente resultado:

Ejemplo 5.51

```
Juan es hermano de Juan Luis
Jose es hermano de Jose Luis
```

slice(inicio [, fin])

Extrae un trozo de una cadena desde inicio hasta fin - 1. Si no se indica fin, se extrae hasta el final de la cadena. Ejemplo:

Ejemplo 5.52

```
var cad = "Juan es hermano de Juan Luis";
var aux;

aux = cad.slice(8);
document.write(aux + "<BR>");
aux = cad.slice(8, 15);
document.write(aux + "<BR>");
```

Este ejemplo produce el siguiente resultado:

Ejemplo 5.53

```
hermano de Juan Luis
hermano
```

split(separador [, limite])

Divide una cadena en función del separador. El resultado de la división se almacena en un array. El valor de limite indica el número máximo de divisiones que se pueden hacer. Ejemplo:

Ejemplo 5.54

```
var cad = "Juan es hermano de Juan Luis";
var aux, i;

aux = cad.split(" ");
for(i = 0; i < aux.length; i++)
document.write("Posicion " + i + ": " + aux[i] + "<BR>");
```

El ejemplo anterior genera el siguiente resultado:

Ejemplo 5.55

```
Posicion 0: Juan
Posicion 1: es
Posicion 2: hermano
Posicion 3: de
Posicion 4: Juan
Posicion 5: Luis
```

5.7. Operaciones matemáticas

JavaScript dispone del objeto Math que posee una serie de propiedades y métodos que permiten trabajar con constantes y funciones matemáticas. Todos los métodos y propiedades son estáticos, por lo que no hace falta crear un objeto de tipo Math para usarlos. En el Cuadro 5.8 se muestran las principales propiedades del objeto Math.

Propiedad	Descripción
E	Constante de Euler, la base de los logaritmos naturales (neperianos). Aproximadamente 2.718
LN10	El logaritmo natural de 10. Aproximadamente 2.302
LN2	El logaritmo natural de 2. Aproximadamente 0.693
LOG10E	El logaritmo decimal (base 10) de E. Aproximadamente 0.434
LOG2E	El logaritmo en base 2 de E. Aproximadamente 1.442
PI	Razón de la circunferencia de un círculo a su diámetro. Aproximadamente 3.141
SQRT1_2	Raíz cuadrada de 1 / 2. Aproximadamente 0.707
SQRT2	Raíz cuadrada de 2. Aproximadamente 1.414

Cuadro 5.8: Propiedades del objeto Math

Los principales métodos del objeto Math son: abs, acos, asin, atan, ceil, cos, sin, tan, exp, floor, log, max, min, pow, random, round y sqrt.

abs

Devuelve el valor absoluto de un número.

Ejemplo 5.56

```
document.write(Math.abs(3.5) + "<BR>");
document.write(Math.abs(2) + "<BR>");
document.write(Math.abs(-0.5) + "<BR>");
document.write(Math.abs(-4) + "<BR>");
```

El código anterior produce la siguiente salida:

Ejemplo 5.57

```
3.5
2
```

```
.5
4
```

acos, asin, atan

Devuelve el arcocoseno, arcoseno y arcotangente en radianes de un número.

ceil

Devuelve el entero menor mayor o igual que un número. Ejemplo:

Ejemplo 5.58

```
document.write("ceil(2): " + Math.ceil(2) + "<BR>");
document.write("ceil(2.0): " + Math.ceil(2.0) + "<BR>");
document.write("ceil(2.05): " + Math.ceil(2.05) + "<BR>");
```

El código anterior genera la siguiente salida:

Ejemplo 5.59

```
ceil(2): 2
ceil(2.0): 2
ceil(2.05): 3
```

cos, sin, tan

Devuelve el coseno, seno y tangente de un ángulo expresado en radianes.

exp

Devuelve E elevado a un número.

floor

Devuelve el mayor entero menor o igual que un número. Ejemplo:

Ejemplo 5.60

```
document.write("floor(2): " + Math.floor(2) + "<BR>");
document.write("floor(2.0): " + Math.floor(2.0) + "<BR>");
document.write("floor(2.05): " + Math.floor(2.05) + "<BR>");
document.write("floor(2.99): " + Math.floor(2.99) + "<BR>");
```

Este ejemplo muestra el siguiente resultado:

Ejemplo 5.61

Ejemplo 5.61

```
floor(2): 2
floor(2.0): 2
floor(2.05): 2
floor(2.99): 2
```

Devuelve el logaritmo natural (en base E) de un número.

max, min

Devuelve el mayor (mínimo) de dos números. Ejemplo:

Ejemplo 5.62

```
document.write("max(3, 5): " + Math.max(3, 5) + "<BR>");
document.write("min(3, 5): " + Math.min(3, 5) + "<BR>");
```

El código anterior genera la siguiente salida:

Ejemplo 5.63

```
max(3, 5): 5
min(3, 5): 3
```

pow

Devuelve la base elevada al exponente. Ejemplo:

Ejemplo 5.64

```
document.write("pow(2, 3): " + Math.pow(2, 3) + "<BR>");
document.write("pow(3, 4): " + Math.pow(3, 4) + "<BR>");
document.write("pow(5, 0): " + Math.pow(5, 0) + "<BR>");
```

El ejemplo anterior genera el siguiente resultado:

Ejemplo 5.65

```
pow(2, 3): 8
pow(3, 4): 81
pow(5, 0): 1
```

random

Produce un número pseudoaleatorio entre 0 y 1. Ejemplo:

Ejemplo 5.66

```
document.write("random(): " + Math.random() + "<BR>");
document.write("random(): " + Math.random() + "<BR>");
```

Este ejemplo produce la siguiente salida:

Ejemplo 5.67

```
random(): .32779162476197754
random(): .8945828404144374
```

round

Devuelve el entero más cercano a un número. Si la parte decimal es menor que 0.5, lo redondea al entero mayor menor o igual que el número; si la parte decimal es igual o mayor que 0.5, lo redondea al entero menor mayor o igual que el número. Ejemplo:

Ejemplo 5.68

```
document.write("round(3.49): " + Math.round(3.49) + "<BR>");
document.write("round(3.5): " + Math.round(3.5) + "<BR>");
document.write("round(3.51): " + Math.round(3.51) + "<BR>");
```

Produce la siguiente salida:

Ejemplo 5.69

```
round(3.49): 3
round(3.5): 4
round(3.51): 4
```

sqrt

Devuelve la raíz cuadrada de un número.

5.8. Validación de formularios

Una de las utilidades más importantes que ofrece *JavaScript* es la posibilidad de realizar validaciones inmediatas de la información introducida por un usuario en un formulario. Aunque existe la alternativa de utilizar un programa **CGI** o un **ASP** que realice la misma función en el servidor, poder llevar a cabo este proceso en el cliente, ahorra tiempo de espera a los usuarios, disminuye el número de conexiones al

servidor y limita la carga del servidor.

Antes de estudiar la validación de formularios, conviene leer el Capítulo 6, ya que en él se explica **DOM**, una representación de los distintos elementos que componen una página web. Mediante **DOM** se accede a los formularios y sus controles.

5.8.1. Validación campo nulo

Muchas veces interesa comprobar si se ha introducido información en un campo: que no se deje en blanco algún dato. En el siguiente ejemplo, la función compruebaVacio determina si en un campo se ha introducido información, teniendo en cuenta que se considera información todo lo que sea distinto de vacío, espacios en blanco o tabuladores. En la Figura 5.2 vemos como se visualiza el siguiente código en un navegador.

Ejemplo 5.70

```
<HTML>
<HEAD>
<TITLE>Validación de un campo vacío</TITLE>
<SCRIPT LANGUAGE="JavaScript">
error = new creaError();
errores = new Array();
errores[0] = "Campo vacío";
errores[1] = "Campo sólo contiene blancos o tabuladores";

function creaError()
{
	this.valor = 0;
	return this;
}

function compruebaVacio(contenido, error)
{
	if(contenido.length == 0)
	{
		error.valor = 0;
		error.posicion = 0;
		return true;
	}
	for(var i = 0; i < contenido.length; i++)
		if(contenido.charAt(i) != ' ' &&
				contenido.charAt(i) != '\t')
			return false;

	error.valor = 1;
	return true;
}
```

```
function valida()
{
        if(compruebaVacio(document.formulario.campo.value, error))
                alert("El campo no es correcto: " + errores[error.valor]);
        else
                alert("El campo es correcto: Campo no vacío");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="formulario">
<B><CENTER>Validación de un campo vacío</CENTER></B>
<BR>
Campo: <INPUT TYPE="TEXT" NAME="campo">
<INPUT TYPE="BUTTON" VALUE="Valida" ONCLICK="valida()">
</FORM>
</BODY>
</HTML>
```



Figura 5.2: Validación campo nulo

5.8.2. Validación alfabética

Un tipo elemental de información que aparece en muchos formularios es el tipo alfabético compuesto por los caracteres alfabéticos del idioma en particular con el que se trabaje. En el siguiente ejemplo, las funciones compruebaAlfa, compruebaAlfaMin y compruebaAlfaMay validan que un valor sea una palabra, una palabra en minúsculas y una palabra en mayúsculas, incluyendo la posibilidad de que existan letras acentuadas o con diéresis. En la Figura 5.3 vemos el aspecto de la página visualizada en un navegador.

Ejemplo 5.71

```
<HTML>
<HEAD>
<TITLE>Validación alfabética de un campo</TITLE>
<SCRIPT LANGUAGE="JavaScript">
mayusculas = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZÁÉÍÓÚÜ";
minusculas = "abcdefghijklmnñopqrstuvwxyzáéíóúü";

error = new creaError();
errores = new Array();
errores[1] = "Falta carácter alfabético";
errores[2] = "Falta carácter alfabético en minúsculas";
errores[3] = "Falta carácter alfabético en mayúsculas";

function creaError()
{
        this.valor = 0;
        this.posicion = 0;
        return this;
}

function esMinuscula(c)
{
        return minusculas.indexOf(c) >= 0;
}


function esMayuscula(c)
{
        return mayusculas.indexOf(c) >= 0;
}

function esLetra(c)
{
        return esMinuscula(c) || esMayuscula(c);
}

function compruebaAlfa(contenido, error)
{
        for(var i = 0; i < contenido.length; i++)
                if(!esLetra(contenido.charAt(i)))
                {
                        error.valor = 1;
                        error.posicion = i + 1;
                        return false;
                }

        return true;
}

function compruebaAlfaMin(contenido, error)
{
        for(var i = 0; i < contenido.length; i++)
```

```
			if(!esMinuscula(contenido.charAt(i)))
			{
				error.valor = 2;
				error.posicion = i + 1;
				return false;
			}

		return true;
}

function compruebaAlfaMay(contenido, error)
{
		for(var i = 0; i < contenido.length; i++)
			if(!esMayuscula(contenido.charAt(i)))
			{
				error.valor = 3;
				error.posicion = i + 1;
				return false;
			}

		return true;
}

function valida(valor)
{
		var correcto;
		var contenido = document.formulario.campo.value;

		switch(valor)
		{
			case 1:
				correcto = compruebaAlfa(contenido, error);
				break;

			case 2:
				correcto = compruebaAlfaMin(contenido, error);
				break;

			case 3:
				correcto = compruebaAlfaMay(contenido, error);
				break;

			default:
				break;
		}

		if(correcto)
			alert("El campo es válido");
		else
			alert("El campo no es válido: " + errores[error.valor] +
						" en la posición " + error.posicion);
}
</SCRIPT>
```

```
<BODY>
<FORM NAME="formulario">
<B><CENTER>Validación alfabética de un campo</CENTER></B>
<BR>
Campo: <INPUT TYPE="TEXT" NAME="campo">
<INPUT TYPE="BUTTON" VALUE="Palabra" ONCLICK="valida(1)">
<INPUT TYPE="BUTTON" VALUE="Minúsculas" ONCLICK="valida(2)">
<INPUT TYPE="BUTTON" VALUE="Mayúsculas" ONCLICK="valida(3)">
</FORM>
</BODY>
</HTML>
```

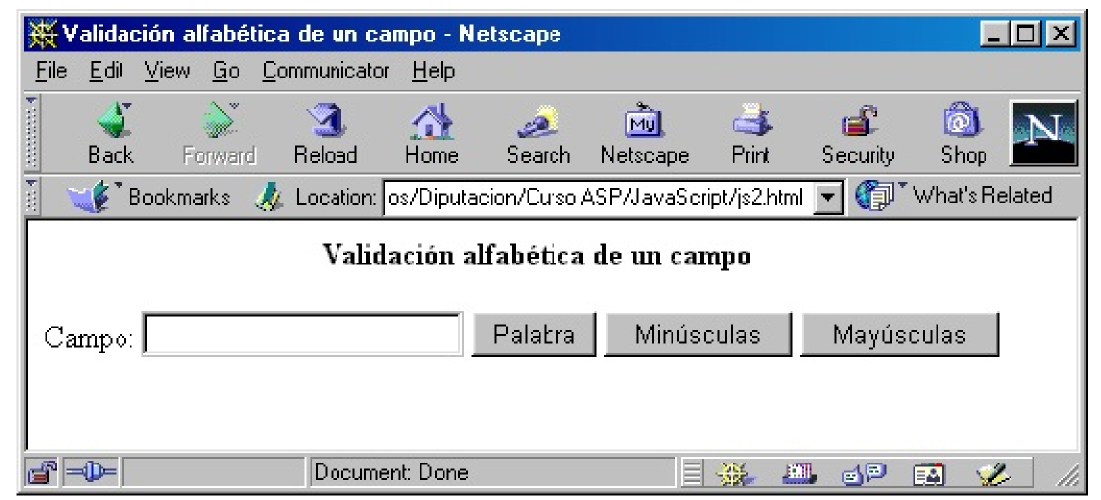


Figura 5.3: Validación alfabética

5.8.3. Validación numérica

Se pueden distinguir tres tipos de formatos numéricos básicos:

- Número natural: formado por los caracteres numéricos.
- Número entero: formado por un signo inicial opcional (+, -), seguido de un número natural.
- Número real: formado por un signo inicial opcional (+, -), seguido de caracteres numéricos y, opcionalmente, seguido de la coma decimal y otra serie de caracteres numéricos.

En el siguiente ejemplo, las funciones compruebaNatural, compruebaEntero y compruebaReal validan que un valor sea un número natural, un número entero y un número real‹ 34 › respectivamente. En la Figura 5.4 vemos el aspecto de la página visualizada en un navegador.

Ejemplo 5.72

```
<HTML>
<HEAD>
<TITLE>Validación numérica de un campo</TITLE>
<SCRIPT LANGUAGE="JavaScript">
numeros = "0123456789";
puntoDecimal = ".,";
signos = "+-";

error = new creaError();
errores = new Array();
errores[1] = "Campo vacío no contiene ningún valor";
errores[2] = "Carácter ilegal en un número";
errores[3] = "Carácter ilegal";
errores[4] = "Sólo ha insertado un signo";
errores[5] = "Parte decimal vacía";

function creaError()
{
    this.valor = 0;
    this.posicion = 0;
    return this;
}

function numero(c)
{
    return numeros.indexOf(c) >= 0;
}

function signo(c)
{
    return signos.indexOf(c) >= 0;
}

function compruebaNatural(contenido, error)
{
    if(contenido.length == 0)
    {
        error.valor = 1;
        error.posicion = 1;
        return false;
    }

    for(var i = 0; i < contenido.length; i++)
        if(!numero(contenido.charAt(i)))
        {
            error.valor = 2;
            error.posicion = i + 1;
            return false;
        }

    return true;
}
```

```
function signoCorrecto(contenido, error)
{
	if(contenido.length == 0)
	{
		error.valor = 1;
		error.posicion = 1;
		return false;
	}

	if(!numero(contenido.charAt(0)) &&
			!signo(contenido.charAt(0)))
	{
		error.valor = 3;
		error.posicion = 1;
		return false;
	}

	return true;
}

function compruebaEntero(contenido, error)
{
	if(!signoCorrecto(contenido, error))
		return false;

	if(numero(contenido.charAt(0)))
		var aux = compruebaNatural(contenido, error);
	else
	{
		var aux = compruebaNatural(contenido.substring(1,
				contenido.length), error);
		if(!aux)
			error.posicion++;
		if(error.valor == 1)
		{
			error.valor = 4;
			error.posicion = 1;
		}
	}

	return aux;
}

function compruebaReal(contenido, error)
{
	var aux = compruebaEntero(contenido, error);
	var posicion = error.posicion - 1;

	if(!aux && error.valor == 2 &
			puntoDecimal.indexOf(contenido.charAt(posicion)) >= 0)
	{
		var aux = compruebaNatural(contenido.substring(
```

```
			error.posicion, contenido.length), error);
			if(!aux && error.valor == 1)
			{
				error.valor = 5;
			}
			if(!aux)
			error.posicion += posicion + 1;
		}

		return aux;
}

function valida(valor)
{
		var correcto;
		var contenido = document.formulario.campo.value;

		switch(valor)
		{
			case 1:
				correcto = compruebaNatural(contenido, error);
				break;

			case 2:
				correcto = compruebaEntero(contenido, error);
				break;

			case 3:
				correcto = compruebaReal(contenido, error);
				break;

			default:
				break;
		}

		if(correcto)
			alert("El campo es válido");
		else
			alert("El campo no es válido: " + errores[error.valor] +
					" en la posición " + error.posicion);
}
</SCRIPT>
<BODY>
<FORM NAME="formulario">
<B><CENTER>Validación numérica de un campo</CENTER></B>
<BR>
Campo: <INPUT TYPE="TEXT" NAME="campo">
<INPUT TYPE="BUTTON" VALUE="Natural" ONCLICK="valida(1)">
<INPUT TYPE="BUTTON" VALUE="Entero" ONCLICK="valida(2)">
<INPUT TYPE="BUTTON" VALUE="Real" ONCLICK="valida(3)">
</FORM>
</BODY>
</HTML>
```

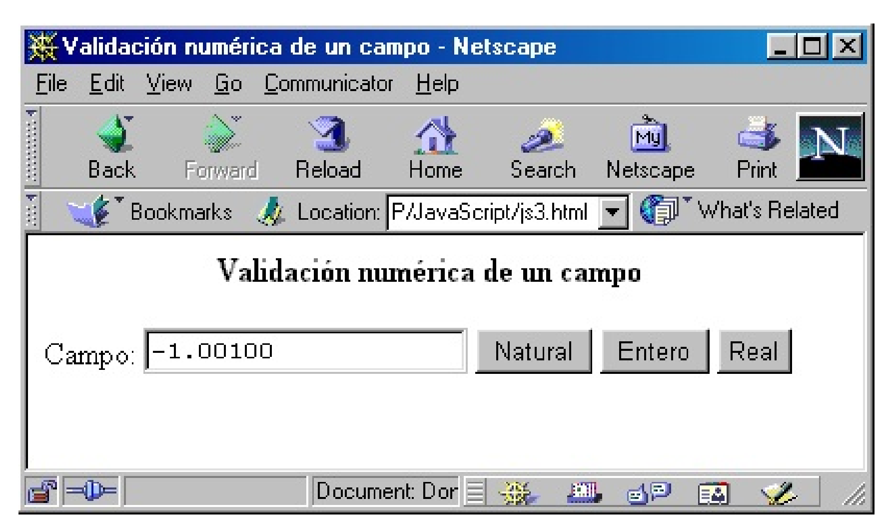


Figura 5.4: Validación numérica

Capítulo 6

Modelo de objetos de documento

El Modelo de Objetos de Documento es un interfaz que permite acceder y modificar la estructura y contenido de una página web. Especifica como se puede acceder a los distintos elementos (enlaces, imágenes, formularios, etc.) de una página y como se pueden modificar. En este capítulo vamos a describir los objetos que componen este modelo, sus propiedades, métodos y eventos.

Índice General

6.1. Introducción

El Modelo de Objetos de Documento **DOM** se suele confundir con los lenguajes de *script*: se piensa que forma parte de ellos, cuando en realidad son independientes uno del otro.

El modelo de objetos permite acceder a los elementos **HTML** de un documento desde los lenguajes de *script*. Para lograrlo, se crean una serie de objetos que representan (exponen) dichos elementos. Estos objetos guardan entre ellos unas relaciones de parentesco (se establece una jerarquía) que refleja la lógica de una página **HTML**: una página se presenta en una ventana, que posee un documento, el cual a su vez puede tener una serie de formularios, que pueden contener elementos como botones, cuadros de texto, etc., los cuales tienen a su vez una serie de propiedades, etc.

El modelo de objetos se modifica de una versión de navegador a otra, añadiéndose nuevos objetos y modificándose la jerarquía existente entre ellos. Al igual que ocurre con los lenguajes de *script*, cada navegador posee su propio modelo de objetos, que coinciden entre ellos en algunos objetos, propiedades y métodos. Existe un intento de estandarización por parte de **W3C** del modelo de objetos de documento.

6.2. Modelo de objetos en Netscape Communicator

Cuando se carga un objeto en el navegador, se crean una serie de objetos *JavaScript* llamados objetos del navegador (*Navigator objects*) con propiedades basadas en el documento **HTML** que se está mostrando. Estos objetos poseen una jerarquía que refleja la estructura de la página **HTML**. En la Figura 6.1 vemos la jerarquía del modelo de objetos en Netscape Communicator 4.0 y superiores (en versiones superiores puede ser que se vea ampliado).

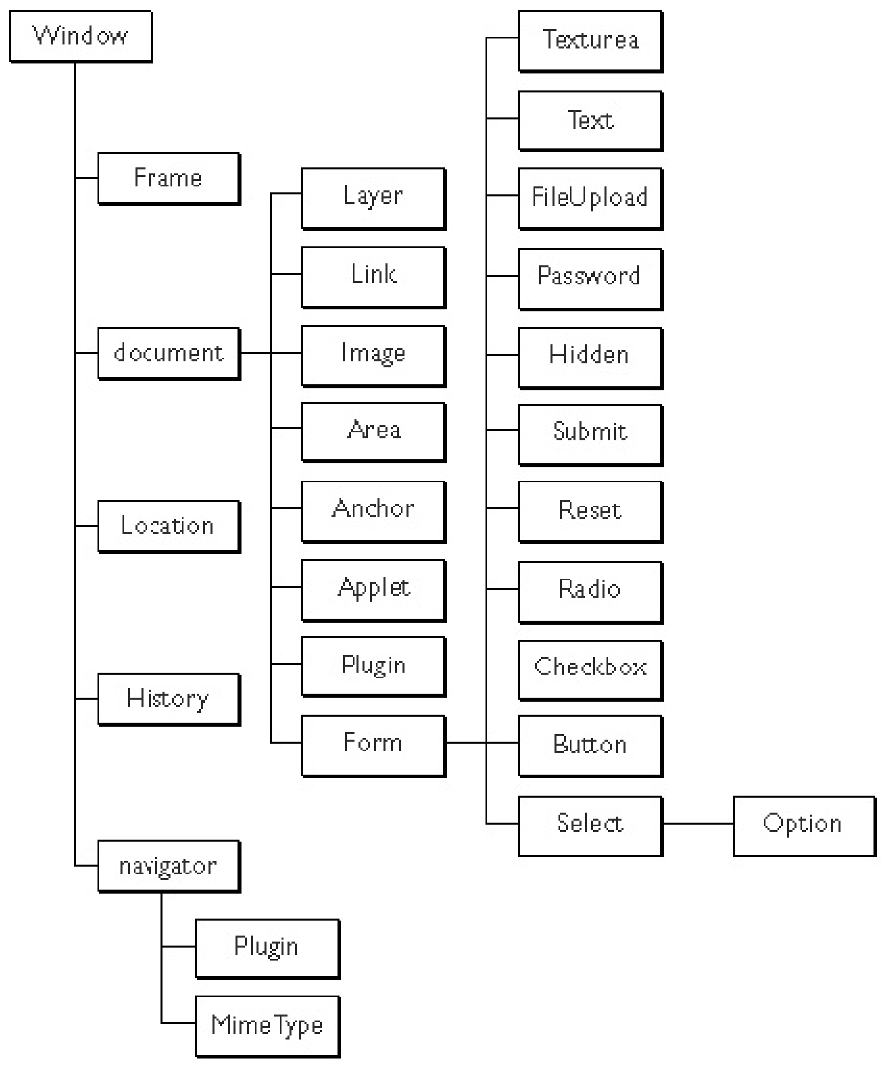


Figura 6.1: Modelo de objetos en Netscape Communicator

En esta jerarquía, los descendiente de un objeto se consideran propiedades de dicho objeto, aunque por sí mismos también son objetos. Por ejemplo, un formulario llamado form1 es un objeto y también una propiedad del objeto document y se accede como document.form1.

Como todos los objetos cuelgan del objeto window, éste se pude obviar y no escribir cuando se accede a un objeto de la ventana actual[35]. Así, window.navigator.appName y navigator.appName representan la misma propiedad y las sentencias window.document.writeln() y document.writeln() llaman al mismo método.

Cualquier página posee los siguientes objetos:

- document: contiene una serie de propiedades basadas en el contenido del documento, como su título, su color de fondo, sus enlaces y formularios.
- history: contiene una serie de propiedades que representan las URLs que el cliente ha visitado previamente (contiene el historial de navegación).
- location: posee propiedades basadas en la **URL** actual.
- navigator: posee propiedades que representan el nombre y la versión de navegador que se está usando, propiedades para los tipos *Multipurpose Internet Mail Extensions* (**MIME**) que acepta el cliente y para los *plug-ins* que están instalados en el cliente.
- window: se trata del objeto de más alto nivel; tiene propiedades que se aplican a la ventana completa. Cada ventana hija en un documento dividido en marcos (*frames*) posee su propio objeto window.

Dependiendo del contenido de la página, el documento puede poseer otra serie de objetos. Por ejemplo, un formulario (definido mediante la etiqueta <FORM> ... </FORM>) en el documento tiene su correspondiente objeto form en la jerarquía de objetos.

6.2.1. Objeto document

Posee información sobre el documento actual y posee métodos que permiten mostrar una salida en formato **HTML** al usuario. Se crea un objeto document por cada etiqueta <BODY> ... </BODY>.

Propiedades

alinkColor Atributo ALINK de la etiqueta BODY.

anchors Array que contiene una entrada para cada ancla del documento.

applets Array que contiene una entrada por cada *applet* en el documento.

bgColor Atributo BGCOLOR de la etiqueta BODY.

classes Crea un objeto Style que se usa para especificar el estilo de las etiquetas **HTML**.

cookie Especifica una *cookie*.

domain Nombre de dominio del servidor que sirvió el documento.

embeds Array que contiene una entrada por cada *plug-in* del documento.

fgColor Atributo TEXT de la etiqueta BODY.

formName Una propiedad por cada formulario con nombre en el documento.

forms Array que contiene una entrada por cada formulario en el documento.

height Altura del documento en pixels.

ids Crea un objeto Style que permite especificar el estilo de cada etiqueta **HTML**.

images Array que contiene una entrada para cada imagen del documento.

lastModified Fecha en la que el documento se modificó por última vez.

layers Array que contiene un entrada por cada capa (*layer*) en el documento.

linkColor Atributo LINK de la etiqueta BODY.

links Array que contiene una entrada por cada enlace en el documento.

plug-ins Array que contiene una entrada por cada *plug-in* en el documento.

referrer Especifica la **URL** del documento que llamó al actual.

tags Crea un objeto Style que permite especificar los estilos de las etiquetas **HTML**.

title Contenido de la etiqueta TITLE de la sección HEAD.

URL URL completa del documento.

vlinkColor Atributo VLINK de la etiqueta BODY.

width Anchura del documento en pixels.

Métodos

captureEvents Establece que el documento capture todos los eventos del tipo especificado.

close Cierra un flujo de salida y fuerza que la información se muestre.

contextual Especifica un objeto Style que puede fijar el estilo de las etiquetas HTML.

getSelection Devuelve una cadena que contiene el texto seleccionado.

handleEvent Invoca el manejador del evento especificado.

open Abre un flujo que recibirá la salida de los métodos write y writeln.

releaseEvents Establece que el documento no capture los eventos del tipo especificado.

routeEvent Pasa un evento a lo largo de la jerarquía de eventos.

write Escribe una o más expresiones **HTML** en el documento.

writeln Escribe una o más expresiones **HTML** en el documento y finaliza con un salto de línea.

Eventos

onClick Se produce cuando un objeto de un formulario o un enlace es pulsado con el botón del ratón.

onDblClick Se produce cuando un objeto de un formulario o un enlace es pulsado dos veces con el botón del ratón.

onKeyDown Se produce cuando el usuario pulsa una tecla.

onKeyPress Se produce cuando el usuario presiona o mantiene pulsada una tecla.

onKeyUp Se produce cuando el usuario deja de pulsar una tecla.

onMouseDown Se produce cuando el usuario pulsa un botón del ratón.

onMouseUp Se produce cuando el usuario deja de pulsar un botón del ratón.

El siguiente código de ejemplo muestra los enlaces que posee un documento.

Ejemplo 6.1

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
</HEAD>
<BODY>
<A HREF="pag1.html">Enlace a la pagina 5</A>
<BR>
<A HREF="pag2.html">Enlace a la pagina 2</A>
<BR>
<A HREF="pag3.html">Enlace a la pagina 3</A>
<BR><BR>
<SCRIPT LANGUAGE="JavaScript">
document.write("Hay " + document.links.length + " enlaces<BR>\n");

for(i = 0; i < document.links.length; i++)
{
        document.write(document.links[i]);
        document.write("<BR>\n");
}
</SCRIPT>
</BODY>
</HTML>
```

El siguiente ejemplo muestra el valor de los atributos BGCOLOR, TEXT, LINK, ALINK y VLINK. Además, también aparece la fecha de la última modificación (lastModified). En la Figura 6.2 vemos como se visualiza esta página en un navegador.

Ejemplo 6.2

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE" TEXT="BLACK" LINK="NAVY"
                ALINK="BLUE" VLINK="RED">
<SCRIPT LANGUAGE="JavaScript">
document.write("BGCOLOR = " + document.bgColor + "<BR>\n");
document.write("TEXT = " + document.fgColor + "<BR>\n");
document.write("LINK = " + document.linkColor + "<BR>\n");
document.write("ALINK = " + document.alinkColor + "<BR>\n");
document.write("VLINK = " + document.vlinkColor + "<BR>\n");
```

```
document.write("<BR>\n");
document.write("lastModified = " + document.lastModified + "<BR>\n");
</SCRIPT>
</BODY>
</HTML>
```

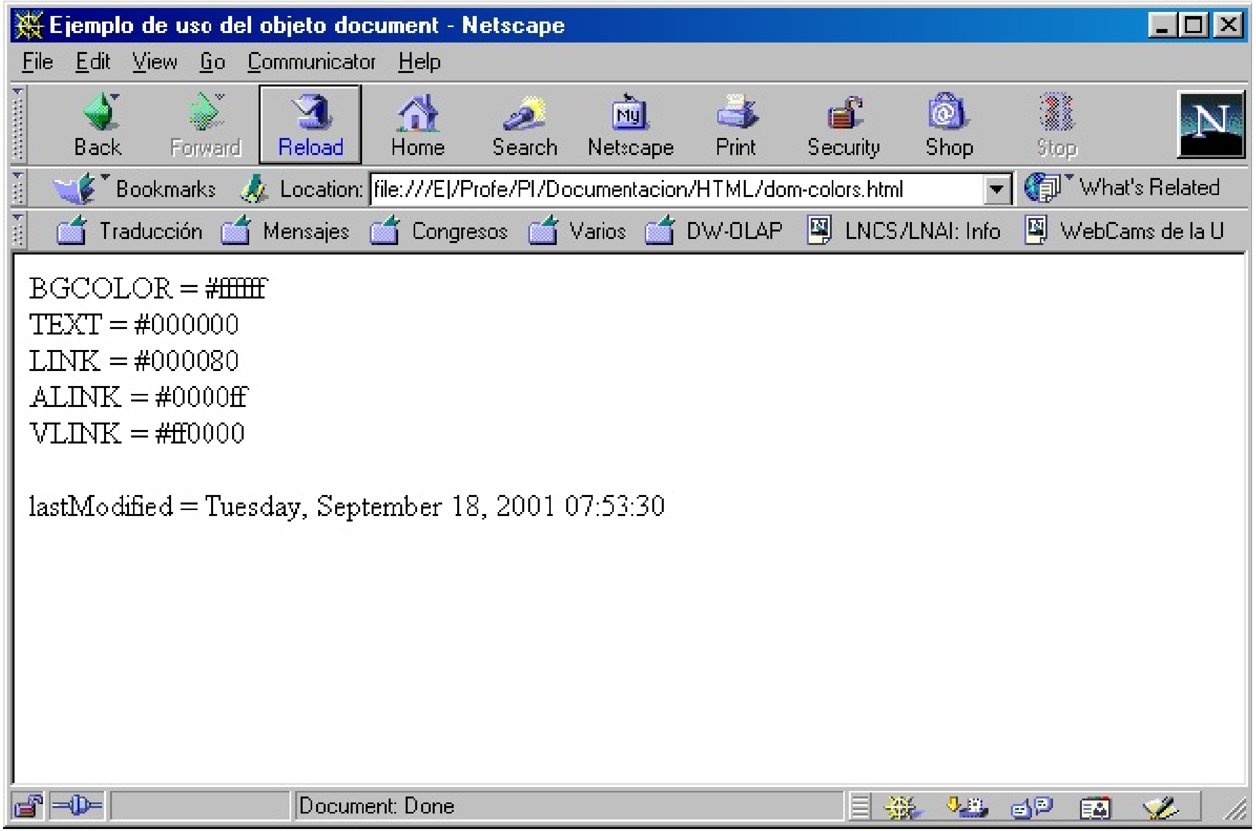


Figura 6.2: Propiedades del objeto document

6.2.2. Cómo acceder a los controles de un formulario

Un documento (document) puede poseer varios formularios, cada uno con sus correspondientes controles. Para acceder a los controles de un formulario se tiene que recorrer la jerarquía que se puede ver en la Figura 6.1: objeto window‹ 36 ›, objeto document, nombre del formulario‹ 37 ›, nombre del control y propiedad del control.

Por ejemplo, si tenemos el siguiente formulario:

Ejemplo 6.3

```
<FORM NAME="miForm">
Nombre: <INPUT TYPE="TEXT" NAME="nombre">
</FORM>
```

para acceder al valor (propiedad value) que contiene el campo de texto nombre se

tiene que escribir:

Ejemplo 6.4

```
document.miForm.nombre.value
```

Hay que prestar mucha atención a las mayúsculas y minúsculas, ya que como se explicó en el Capítulo 5, *JavaScript* es un lenguaje sensible a minúsculas/- mayúsculas.

Cada uno de los controles de un formulario tiene una serie de propiedades, métodos y eventos. Todos los controles poseen el método focus() que permite fijar el foco sobre el objeto.

A continuación mostramos las propiedades más importantes de los controles más usados.

Campos de verificación

Los campos de verificación poseen las siguiente propiedades:

checked Valor booleano que indica si se encuentra seleccionado.

defaultChecked Valor booleano que indica si por defecto se encuentra seleccionado.

value Valor asociado con el control.

Por ejemplo, el siguiente código muestra en una ventana el estado de los campos de verificación de un formulario.

Ejemplo 6.5

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function valida()
{
	if(document.miForm.ingles.checked)
		alert("Inglés: SÍ");
	else
		alert("Inglés: NO");
	if(document.miForm.aleman.checked)
		alert("Alemán: SÍ");
	else
		alert("Alemán: NO");
	if(document.miForm.frances.checked)
		alert("Francés: SÍ");
	else
		alert("Francés: NO");
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<FORM NAME="miForm">
Idiomas:
<BR>
Inglés <INPUT TYPE="CHECKBOX" NAME="ingles" VALUE="ing" CHECKED>
Alemán <INPUT TYPE="CHECKBOX" NAME="aleman" VALUE="ale">
Francés <INPUT TYPE="CHECKBOX" NAME="frances" VALUE="fra">
<BR>
<INPUT TYPE="BUTTON" VALUE="Comprobar" ONCLICK="valida()">
</FORM>
</BODY>
</HTML>
```

Campos excluyentes

Posee las mismas propiedades que los campos de verificación:

defaultChecked Valor booleano que indica si por defecto se encuentra seleccionado.

value Valor asociado con el control, que se devuelve al servidor cuando el formulario se envía.

Campos de texto y áreas de texto

Ambos controles poseen estas dos propiedades:

defaultValue Valor por defecto del control.

value Valor actual de control, que se devuelve al servidor cuando el formulario se envía.

Al final del Capítulo 5, cuando se explica la validación de formularios, se pueden ver varios ejemplos donde se hace uso de la propiedad value.

Listas de selección

Las listas de selección poseen las siguientes propiedades:

length Número de opciones en la lista.

options Array donde cada posición representa una opción (objeto option) de la lista.

selectedIndex Índice de la primera opción seleccionada, empezando desde 0. Si ninguna opción está seleccionada, vale -1.

Además, si se quiere obtener el texto y el valor asociado a una opción, se tienen que emplear las propiedades del objeto option:

selected Valor booleano que indica si la opción se encuentra seleccionada.

text Texto de la opción que se muestra en la lista.

value Valor asociado con la opción, que se devuelve al servidor cuando la opción está seleccionada.

6.2.3. Objeto history

Contiene un array que almacena las URLs que el usuario ha visitado en la ventana actual. Mediante los métodos que posee este objeto se puede navegar adelante o atrás en el historial.

Propiedades

current Especifica la **URL** actual del historial.

length Indica el número de entradas que almacena el historial.

next Especifica la **URL** de la siguiente entrada en el historial.

previous Especifica la **URL** de la entrada previa en el historial.

Métodos

back Carga la **URL** previa del historial.

forward Carga la siguiente **URL** del historial.

go Carga la **URL** indicada en el historial.

6.2.4. Objeto location

Representa la **URL** completa asociada a un objeto window. Cada una de las propiedades de este objeto representa una porción de la **URL**. Se accede mediante la propiedad location de un objeto window.

Una **URL** se compone de las siguientes partes:

Ejemplo 6.6

```
protocol://host:port/pathname#hash?search
```

Propiedades

hash Especifica el valor de un ancla en una **URL**.

host Especifica el nombre de dominio o dirección IP de la **URL**.

hostname Especifica la parte host:port de la **URL**.

href Especifica la **URL** entera.

pathname Especifica la ruta de la **URL**.

port Especifica el puerto de comunicaciones que usa el servidor.

protocol Especifica el protocolo de la **URL**.

search Especifica la consulta incluida en la **URL**.

Métodos

reload Recarga el documento actual de la ventana.

replace Carga la **URL** especificada sobre la entrada actual del historial de navegación.

El siguiente ejemplo muestra todas las propiedades del objeto location. En la Figura 6.3 vemos el resultado de visualizar la siguiente página en un navegador.

Ejemplo 6.7

```
<HTML>
<HEAD>
<TILLE>Ejemplo de uso del objeto location</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.writeln("location.hash = " + location.hash + "<BR>");
document.writeln("location.host = " + location.host + "<BR>");
document.writeln("location.hostname = " + location.hostname + "<BR>");
document.writeln("location.href = " + location.href + "<BR>");
document.writeln("location.pathname = " + location.pathname + "<BR>");
document.writeln("location.port = " + location.port + "<BR>");
document.writeln("location.protocol = " + location.protocol + "<BR>");
document.writeln("location.search = " + location.search + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

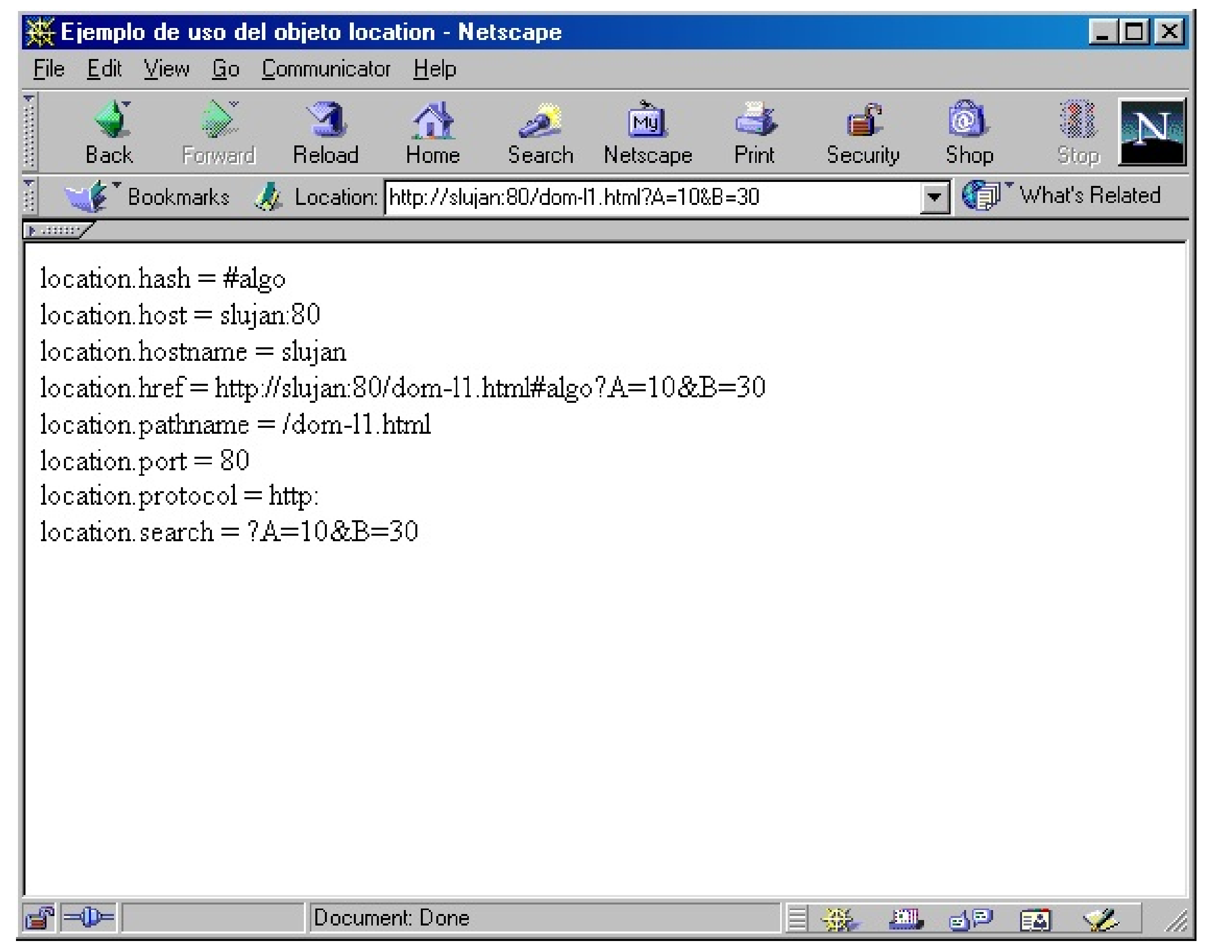


Figura 6.3: Propiedades del objeto location

6.2.5. Objeto navigator

Contiene información sobre la versión del navegador que se está empleando. Todas las propiedades de este objeto son de solo lectura.

Propiedades

appCodeName Nombre en clave del navegador.

appName Nombre del navegador.

appVersion Versión del navegador.

language Idioma que emplea el navegador.

mimeTypes Array que contiene todos los tipos **MIME** que soporta el navegador.

platform Indica el tipo de máquina para la que se compiló el navegador.

plug-ins Array que contiene todos los *plug-ins* instalados.

userAgent Especifica la cabecera **HTTP** user-agent.

Métodos

javaEnabled Chequea si la opción *Java* está activada.

plug-ins.refresh Recarga los documentos que contienen *plug-ins*.

preference Permite a un *script* firmado acceder (lectura y escritura) a ciertas preferencias del navegador.

savePreferences Guarda las preferencias del navegador en prefs.js (fichero de preferencias que posee cada usuario).

taintEnabled Especifica si *data tainting* está activo.

El siguiente ejemplo muestra todas las propiedades del objeto navigator. En la Figura 6.4 vemos el resultado de este ejemplo.

Ejemplo 6.8

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto location</TITLE>
</HEAD>
<BODY>
```

```
<SCRIPT LANGUAGE="JavaScript">
document.writeln("navigator.appCodeName = " + navigator.appCodeName);
document.writeln("<BR>");
document.writeln("navigator.appName = " + navigator.appName);
document.writeln("<BR>");
document.writeln("navigator.appVersion = " + navigator.appVersion);
document.writeln("<BR>");
document.writeln("navigator.language = " + navigator.language);
document.writeln("<BR>");
document.writeln("navigator.mimeTypes = " + navigator.mimeTypes);
document.writeln("<BR>");
document.writeln("navigator.platform = " + navigator.platform);
document.writeln("<BR>");
document.writeln("navigator.plug-ins = " + navigator.plug-ins);
document.writeln("<BR>");
document.writeln("navigator.userAgent = " + navigator.userAgent);
document.writeln("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

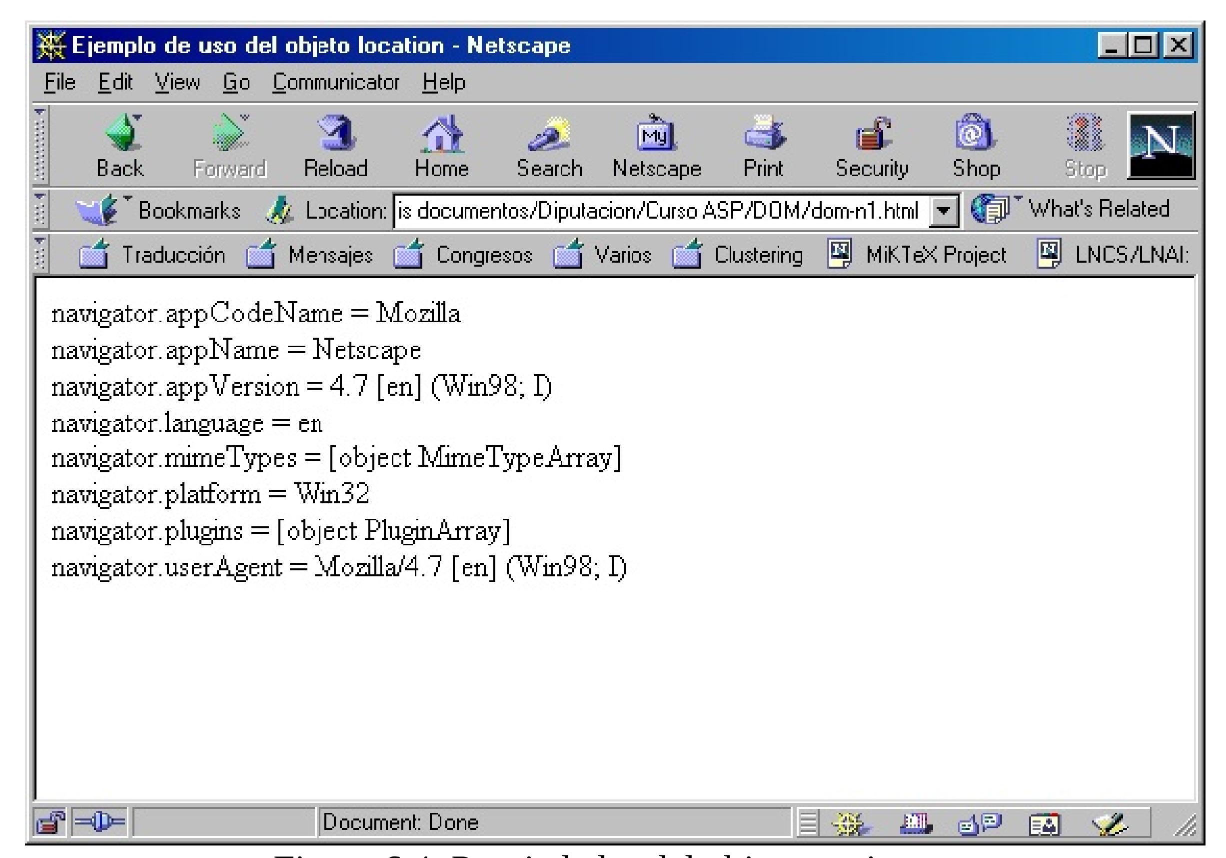


Figura 6.4: Propiedades del objeto navigator

6.2.6. Objeto window

Representa una ventana o un marco del navegador. Es el objeto en la posición superior en la jerarquía de objetos, por encima de todos los demás. Se crea un objeto window por cada etiqueta <BODY> ... </BODY> o <FRAMESET> ... <FRAMESET>. También se crea un objeto window para representar cada marco definido con una etiqueta <FRAME>. Además, también se pueden crear otras ventanas llamando al método window.open.

Propiedades

closed Especifica si una ventana ha sido cerrada.

crypto Permite acceder a las características de encriptación de Netscape Navigator.

defaultStatus Mensaje mostrado por defecto en la barra de estado.

document Contiene información sobre el documento actual y métodos que permiten mostrar contenido **HTML** al usuario.

frames Array que permite acceder a los marcos que posee una ventana.

history Posee información sobre las URLs que el cliente ha visitado dentro de la ventana.

innerHeight Dimensión vertical, en pixels, del área de contenido de la ventana.

innerWidth Dimensión horizontal, en pixels, del área de contenido de la ventana.

length Número de marcos (*frames*) de la ventana.

location Información sobre la **URL** actual.

locationbar Representa la barra de localización de la ventana del navegador.

menubar Representa la barra de menús de la ventana del navegador.

name Nombre único usado para identificar a la ventana.

offscreenBuffering Especifica si las actualizaciones de la ventana se realizan en un buffer.

opener Nombre de la ventana que ha abierto esta ventana usando el método open.

outerHeight Dimensión vertical, en pixels, del límite exterior de la ventana.

outerWidth Dimensión horizontal, en pixels, del límite exterior de la ventana.

pageXOffset Posición actual sobre el eje X, en pixels, de una página vista en la ventana.

pageYOffset Posición actual sobre el eje Y, en pixels, de una página vista en la ventana.

parent Ventana o marco que contiene al marco actual.

personalbar Representa la barra personal (también llamada barra de directorios) de la ventana del navegador.

screenX Posición sobre el eje X del extremo izquierdo de la ventana.

screenY Posición sobre el eje Y del extremo superior de la ventana.

scrollbars Representa las barras de desplazamiento de la ventana.

self Sinónimo de la ventana actual.

status Especifica un mensaje en la barra de estado de la ventana.

statusbar Representa la barra de estado de la ventana.

toolbar Representa la barra de herramientas de la ventana del navegador.

top Sinónimo de la ventana más superior en la jerarquía de ventanas.

window Sinónimo de la ventana actual.

Métodos

alert Muestra un cuadro de diálogo de alerta con un mensaje y un botón OK.

atob Decodifica una cadena de información que ha sido codificada usando la codificación *base-64*.

back Deshace la última navegación en la ventana de nivel superior.

blur Elimina el foco del objeto especificado.

btoa Crea una cadena codificada en *base-64*.

captureEvents Configura la ventana o documento para que capture todos los eventos del tipo especificado.

clearInterval Cancela un temporizador (*timeout*) que se había fijado con el método setInterval.

clearTimeout Cancela un temporizador (*timeout*) que se había fijado con el método setTimeout.

close Cierra la ventana.

confirm Muestra un cuadro de diálogo de confirmación con un mensaje y los botones **OK** y **Cancel**.

crypto.random Devuelve una cadena pseudoaleatoria cuya longitud es el número de bytes especificados.

crypto.signText Devuelve una cadena de información codificada que representa un objeto firmado.

disableExternalCapture Desactiva la captura de eventos externos fijado por el método *enableExternalCapture*.

enableExternalCapture Permite que una ventana con marcos capture los eventos en las páginas cargadas desde distintos sitios (servidores).

find Busca la cadena de texto especificada en el contenido de la ventana especificada.

focus Otorga el foco al objeto especificado.

forward Carga la siguiente **URL** en la lista del historial.

handleEvent Invoca el manejador del método especificado.

home Navega a la **URL** especificada en las preferencias del navegador como la página inicial (*home*).

moveBy Mueve la ventana según el desplazamiento indicado.

moveTo Mueve la esquina superior izquierda de la ventana a la posición de la pantalla indicada.

open Abre una nueva ventana del navegador.

print Imprime el contenido de la ventana o marco.

prompt Muestra un cuadro de diálogo de petición de datos con un mensaje y un campo de entrada.

releaseEvents Configura la ventana para que libere todos los eventos capturados del tipo indicado, enviando el evento a los siguientes objetos en la jerarquía de eventos.

resizeBy Redimensiona la ventana moviendo la esquina inferior derecha la cantidad indicada.

resizeTo Redimensiona la ventana según la altura y anchura indicada.

routeEvent Pasa un evento capturado a través de la jerarquía de eventos normal.

scroll Realiza un *scroll* a las coordenadas indicadas.

scrollBy Realiza un *scroll* en el área de visión según la cantidad especificada.

scrollTo Realiza un *scroll* en el área de visión a las coordenadas especificadas.

setHotKeys Activa o desactiva las *hot keys* en una ventana que no posee menú.

setInterval Evalúa una expresión o llama a una función cada vez que transcurre el número de milisegundos indicado.

setResizable Especifica si el usuario puede redimensionar una ventana.

setTimeout Evalúa una expresión o llama a una función una vez que ha transcurrido el número de milisegundos indicado.

setZOptions Especifica el orden sobre el eje Z.

stop Detiene la carga actual de la página.

Eventos

onBlur Se produce cuando un elemento pierde el foco.

onDragDrop Se produce cuando el usuario deja un objeto sobre la ventana del navegador.

onError Se produce cuando la carga de un documento o una imagen produce un error.

onFocus Se produce cuando un elemento recibe el foco.

onLoad Se produce cuando el navegador termina de cargar una ventana.

onMove Se produce cuando se mueve una ventana o marco.

onResize Se produce cuando se redimensiona una ventana o marco.

onUnload Se produce cuando un usuario cierra un documento.

El siguiente ejemplo muestra como se puede establecer una comunicación entre varias ventanas. A partir de un documento, se puede abrir una ventana nueva en la que se puede escribir texto introducido en la primera ventana. En la Figura 6.5 se muestra la ventana principal y la ventana que se crea mediante el método open.

Ejemplo 6.9

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto window</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var win = null;

function abre() {
	win = window.open("", "Ventana",
				"scrollbars=yes,width=175,height=300");
}

function escri() {
	if(win != null)
	{
		win.document.write(document.formulario.algo.value + "<BR>\n");
		win.focus();
	}
}

function cierra() {
	if(win != null)
	{
		win.close();
		win = null;
	}
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="formulario">
```

```
<P>
<INPUT TYPE="BUTTON" VALUE="Abre una ventana" ONCLICK="abre()">
<P>
<INPUT TYPE="BUTTON" VALUE="Escribe en la ventana" ONCLICK="escri()">
<INPUT TYPE="TEXT" NAME="algo">
<P>
<INPUT TYPE="BUTTON" VALUE="Cierra la ventana" ONCLICK="cierra()">
</FORM>
</BODY>
</HTML>
```

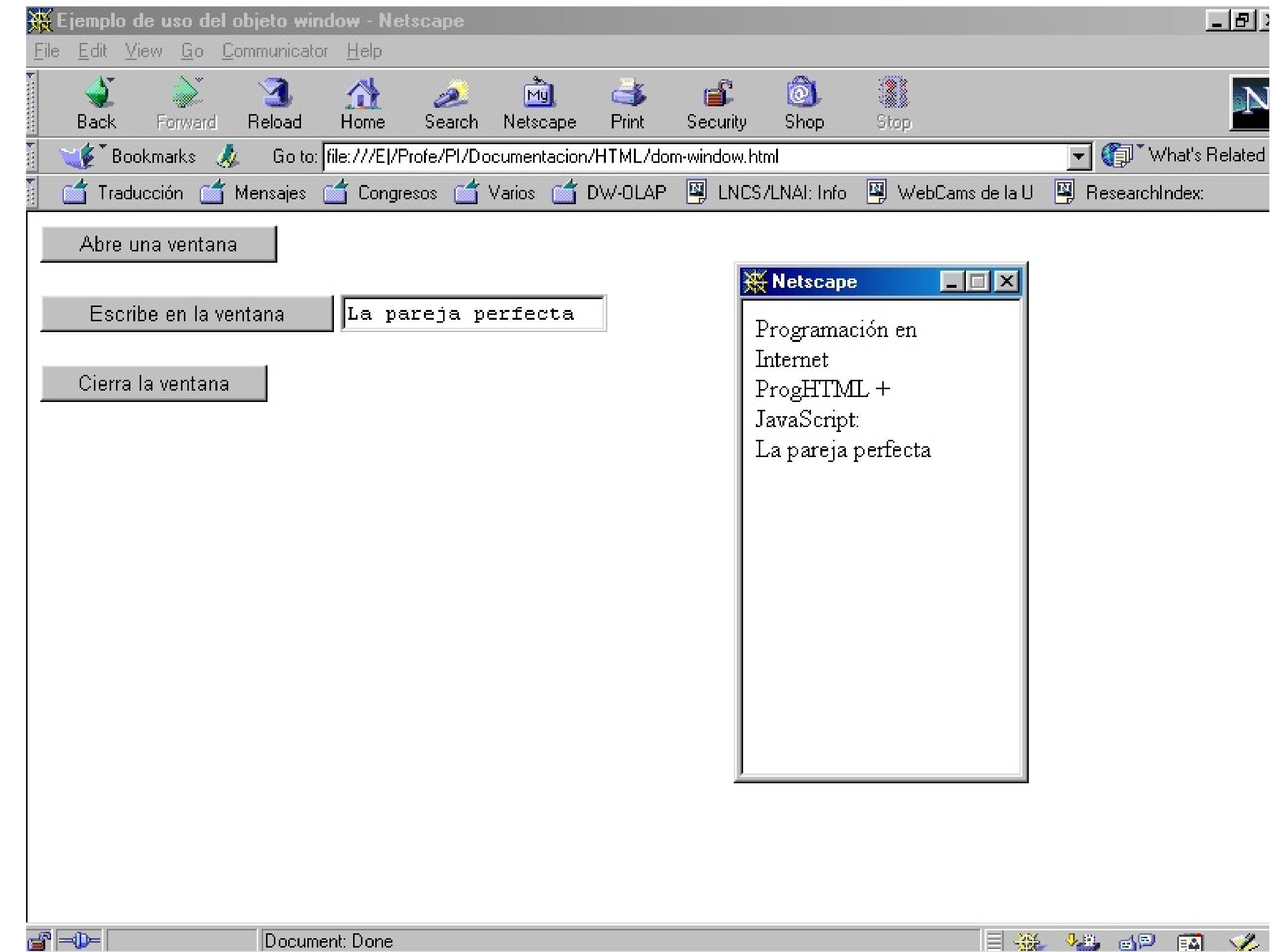


Figura 6.5: Interacción entre varias ventanas a través del objeto window

6.3. Modelo de objetos en Microsoft Internet Explorer

Incluimos en la Figura 6.6 una representación gráfica del modelo de objetos implementado en el navegador de MICROSOFT. Si se compara con el modelo de NETSCAPE (Figura 6.1), se pueden apreciar varias diferencias.

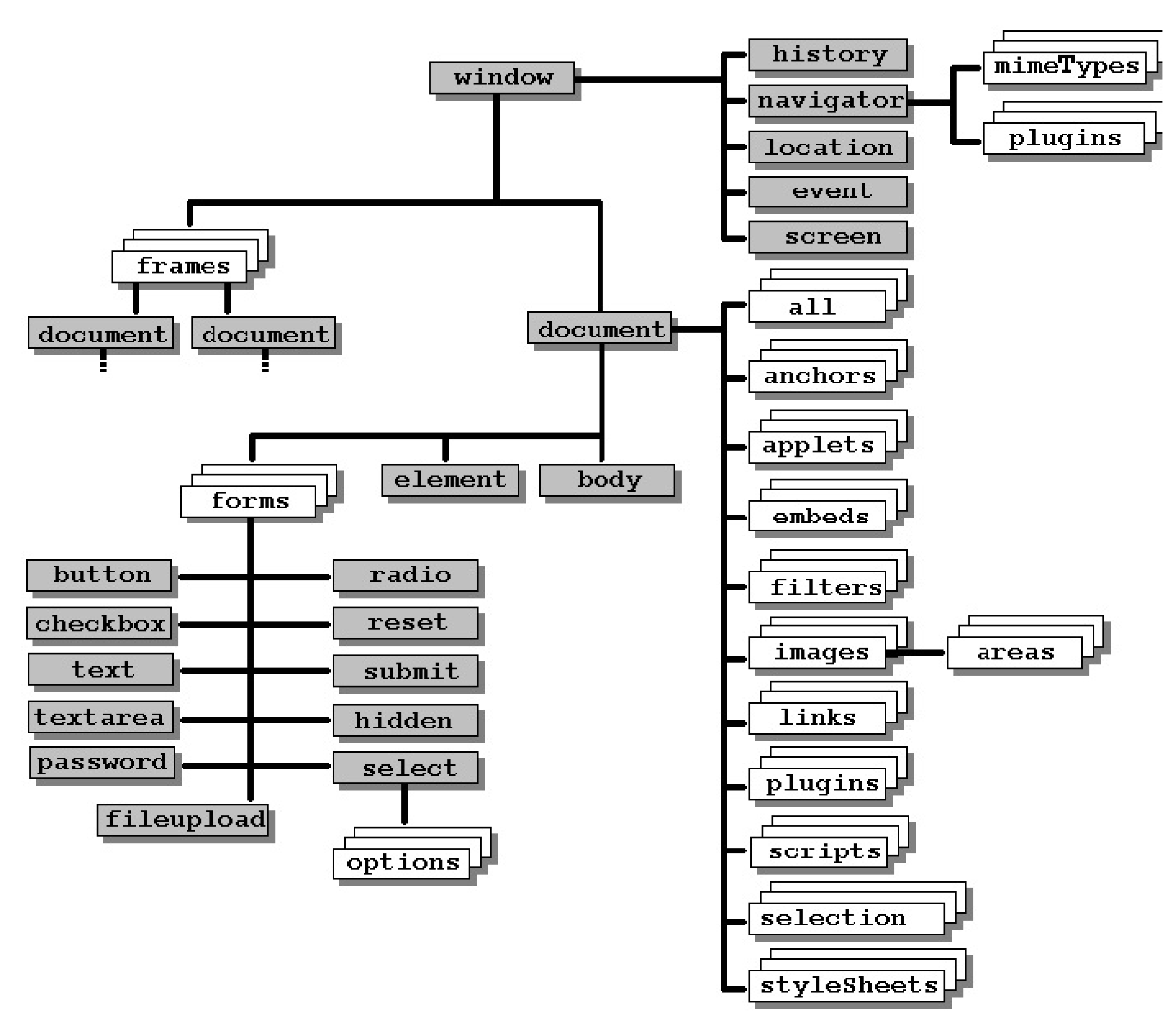


Figura 6.6: Modelo de objetos en Microsoft Internet Explorer

Apéndice A

Resumen etiquetas HTML

En este apéndice se incluye un resumen de la sintaxis de las etiquetas HTML. El objetivo de este apéndice es que sirva como una guía rápida de búsqueda en caso de duda.

Índice General

A.1. Introducción

Este resumen contiene todas las etiquetas **HTML** aceptadas por Netscape Navigator 4.0 y posteriores. Este resumen sólo contiene la sintaxis de las etiquetas. Conforme evoluciona **HTML** aparecen nuevas etiquetas, se añaden atributos nuevos a algunas etiquetas y otras quedan obsoletas (*deprecated*) y se desaconseja su uso.

Cuando una etiqueta se emplea por parejas (inicio y fin), se representa con unos puntos suspensivos, como <HTML> ... </HTML>, mientras que cuando la etiqueta es

individual aparece como <IMG>. Cuando un atributo de una etiqueta puede tomar una serie de posibles valores, estos se han separado con una barra vertical, como por ejemplo ALIGN="LEFT" | "RIGHT" | "CENTER". Otros atributos no reciben ningún valor, como el atributo MULTIPLE de la etiqueta <SELECT> ... </SELECT>.

Las etiquetas se han clasificado en los siguientes grupos:

- Etiquetas que definen la estructura del documento.
- Etiquetas que pueden ir en la cabecera <HEAD>.
- Etiquetas que definen bloques de texto.
- Etiquetas de listas.
- Etiquetas de características del texto.
- Etiquetas de anclas y enlaces.
- Etiquetas de imágenes y mapas de imágenes.
- Etiquetas de tablas.
- Etiquetas de formularios.
- Etiquetas de marcos.
- Etiquetas de situación de contenidos.
- Etiquetas de script.
- Etiquetas de *applets* y *plug-ins*.
- Etiquetas de ajuste del texto.
- Atributos universales.

Para incluir un comentario en una página **HTML** se utiliza la etiqueta <!-- Comentario -->. Se puede emplear para anular una sección de una página, de forma que no se visualice en el navegador, pero sin tener que eliminar dicha sección de la página.

A.2. Etiquetas que definen la estructura del documento

En este grupo se incluyen las etiquetas que definen la estructura básica de un documento **HTML**.

- Etiqueta más externa: <HTML> ... </HTML>.
- Encabezado del documento: <HEAD> ... </HEAD>.
- Contenido principal del documento (cuerpo): <BODY> ... </BODY>. Atributos:
  - BACKGROUND="URL".
  - BGCOLOR="color".
  - TEXT="color".
  - LINK="color".
  - ALINK="color".
  - VLINK="color".
  - ONLOAD="códigoScript".
  - ONUNLOAD="códigoScript".
  - ONBLUR="códigoScript".
  - ONFOCUS="códigoScript".

A.3. Etiquetas que pueden ir en la cabecera

En este grupo se clasifican las etiquetas que pueden usarse en la cabecera de un documento, entre las etiquetas <HEAD> y </HEAD>.

- Título del documento: <TITLE> ... </TITLE>.
- Dirección URL base: <BASE>. Atributos:
  - HREF="URL".
  - TARGET="nombreVentana".
- Información metadocumental: <META>. Atributos:
  - NAME="nombre".
  - HTTP-EQUIV="nombreCampo".
  - CONTENT="valor".

- Definición de estilo: <STYLE> ... </STYLE>. Atributo:
    - TYPE="tipoEstilo".
- Enlace a ficheros externos: <LINK>. Atributos:
    - REL="tipoFichero".
    - TYPE="tipo".
    - SRC="URL".
- Elemento de búsqueda‹ 38 ›: <ISINDEX>. Atributo:
    - PROMPT="texto".
- Código *JavaScript* en el cliente‹ 39 ›: <SCRIPT> ... </SCRIPT>. Atributos:
    - LANGUAGE="nombreLenguajeScript".
    - SRC="URL".

A.4. Etiquetas que definen bloques de texto

En esta sección se incluyen las etiquetas que definen bloques de texto, como encabezados, párrafos, citas, etc.

- Formato dirección: <ADDRESS> ... </ADDRESS>.
- Bloque de texto con sangría: <BLOCKQUOTE> ... </BLOCKQUOTE>.
- Sección de un documento: <DIV> ... </DIV>. Atributo:
    - ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"‹ 40 ›.
- Encabezamientos predefinidos: <H1> ... </H1>, <H2> ... </H2>, <H3> ... </H3>, <H4> ... </H4>, <H5> ... </H5> y <H6> ... </H6>. Atributo:
    - ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY".
- Párrafo: <P> ... </P>. Atributo:
    - ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY".
- Texto preformateado (tipo de letra fija): <PRE> ... </PRE>. Atributos:
    - COLS="columnas".

- WRAP.

- Secuencia literal de caracteres (desactiva intérprete): <XMP> ... </XMP>.

A.5. Etiquetas de listas

En este grupo se encuentran clasificadas las distintas etiquetas que permiten crear listas.

- Lista de directorio: <DIR> ... </DIR>.
- Lista de definición: <DL> ... </DL>. Atributo:
    - COMPACT.
- Término de definición: <DT>.
- Descripción de definición: <DD>.
- Lista de elementos individuales: <MENU> ... </MENU>.
- Lista ordenada: <OL> ... </OL>. Atributos:
    - START="valor".
    - TYPE="A" | "a" | "I" | "i" | "1".
- Lista no ordenada: <UL> ... </UL>. Atributo:
    - TYPE="CIRCLE" | "DISC" | "SQUARE".
- Elemento de una lista: <LI> ... </LI>. Atributos:
    - TYPE="CIRCLE" | "DISC" | "SQUARE" | "A" | "a" | "I" | "i" | "1".
    - VALUE="número".

A.6. Etiquetas de características del texto

En esta sección se muestran las etiquetas que definen las características del texto a nivel de carácter.

- Negrita: <B> ... </B>.

- Tamaño del tipo de letra por defecto: <BASEFONT> ... </BASEFONT>. Atributo:
    - SIZE="número".
- Tamaño de letra mayor: <BIG> ... </BIG>.
- Parpadeante: <BLINK> ... </BLINK>.
- Cita: <CITE> ... </CITE>.
- Código: <CODE> ... </CODE>.
- Enfatizado: <EM> ... </EM>.
- Características del tipo de letra: <FONT> ... </FONT>. Atributos:
    - COLOR="color".
    - FACE="listaTiposDeLetra".
    - POINT-SIZE="tamañoPunto".
    - SIZE="número".
    - WEIGHT="gradoNegrita".
- Cursiva: <I> ... </I>.
- Texto de teclado: <KBD> ... </KBD>.
- Muestra el resto del documento tal cual: <PLAINTEXT>.
- Tipo de letra menor: <SMALL> ... </SMALL>.
- Tachado: <STRIKE> ... </STRIKE> o <S> ... </S>.
- Énfasis fuerte: <STRONG> ... </STRONG>.
- Subíndice: _{...}.
- Superíndice: ^{...}.
- Mecanografiado: <TT> ... </TT>.
- Subrayado: <U> ... </U>.
- Variable: <VAR> ... </VAR>.

A.7. Etiquetas de anclas y enlaces

En esta sección se muestra la etiqueta <A> ... </A>, que se usa tanto para definir las anclas (lugares a donde se puede crear un enlace) como los enlaces.

- Ancla: <A NAME> ... </A>. Atributo:
  - NAME="nombreAncla".
- Enlace: <A HREF> ... </A>. Atributos:
  - HREF="URL".
  - ONCLICK="códigoScript".
  - ONMOUSEOUT="códigoScript".
  - ONMOUSEOVER="códigoScript".
  - TARGET="_blank" | "_parent" | "_self" | "_top" | "nombreVentana".

A.8. Etiquetas de imágenes y mapas de imágenes

En esta sección se muestran las etiquetas que permiten insertar una imagen en un documento **HTML** y crear mapas de imágenes (también llamados mapas o imágenes sensibles).

- Imagen: <IMG>. Atributos:
  - SRC="URL".
  - LOWSRC="URL".
  - ALT="textoAlternativo".
  - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
  - BORDER="anchuraBorde".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - HSPACE="margenHorizontal".
  - VSPACE="margenVertical".
  - ISMAP.

- USEMAP="#nombreMapa".
- NAME="nombreImagen".
- ONABORT="códigoScript".
- ONERROR="códigoScript".
- ONLOAD="códigoScript".
- SUPPRESS="TRUE" | "FALSE".

■ Área de un mapa de imagen: <AREA>. Atributos:

- COORDS="coordenadas".
- SHAPE="CIRCLE" | "RECT" | "POLY".
- HREF="URL".
- NOHREF.
- TARGET="nombreVentana".
- ONMOUSEOUT="códigoScript".
- ONMOUSEOVER="códigoScript".
- NAME="nombreArea".

■ Mapa de imagen: <MAP> ... </MAP>. Atributo:

- NAME="nombreMapa".

A.9. Etiquetas de tablas

Esta sección muestra las etiquetas que se emplean para crear tablas.

■ Tabla: <TABLE> ... </TABLE>. Atributos:

- ALIGN="LEFT" | "CENTER" | "RIGHT".
- BGCOLOR="color".
- BORDER="anchuraBorde".
- CELLPADDING="valor".
- CELLSPACING="valor".
- HEIGHT="altura".
- WIDTH="anchura".

- COLS="numeroDeColumnas".
- HSPACE="margenHorizontal".
- VSPACE="margenVertical".

- Título de la tabla: <CAPTION > ... </CAPTION>. Atributo:
  - ALIGN="BOTTOM" | "TOP".
- Fila de la tabla: <TR> ... </TR>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
  - BGCOLOR="color".
- Celda de una tabla: <TD> ... </TD>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
  - BGCOLOR="color".
  - COLSPAN="valor".
  - ROWSPAN="valor".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - NOWRAP.
- Encabezamiento de una columna o fila: <TH> ... </TH>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
  - BGCOLOR="color".
  - COLSPAN="valor".
  - ROWSPAN="valor".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - NOWRAP.

A.10. Etiquetas de formularios

En este grupo se encuentran las etiquetas que definen los formularios y sus posibles controles.

- Formulario: <FORM> ... </FORM>. Atributos:
  - NAME="nombreFormulario".
  - ACTION="URL".
  - ENCTYPE="tipoCodificación".
  - METHOD="GET" | "POST".
  - TARGET="nombreVentana".
  - ONRESET="códigoScript".
  - ONSUBMIT="códigoScript".
- Botón: <INPUT TYPE="BUTTON">. Atributos:
  - NAME="nombreBotón".
  - VALUE="etiqueta".
  - ONCLICK="códigoScript".
- Casilla de verificación: <INPUT TYPE="CHECKBOX">. Atributos:
  - NAME="nombre".
  - VALUE="valor".
  - CHECKED.
  - ONCLICK="códigoScript".
- Envío de fichero: <INPUT TYPE="FILE">. Atributos:
  - NAME="nombre".
  - VALUE="nombreFichero".
- Elemento oculto: <INPUT TYPE="HIDDEN">. Atributos:
  - NAME="nombre".
  - VALUE="valor".
- Imagen como botón: <INPUT TYPE="IMAGE">. Atributos:
  - NAME="nombre".
  - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".

- SRC="URL".

- Contraseña: <INPUT TYPE="PASSWORD">. Atributos:
  - NAME="nombre".
  - VALUE="texto".
  - MAXLENGTH="máximoNúmeroCaracteres".
  - SIZE="longitudCampo".
  - ONSELECT="códigoScript".
- Botón de radio: <INPUT TYPE="RADIO">. Atributos:
  - NAME="nombre".
  - VALUE="valor".
  - CHECKED.
  - ONCLICK="códigoScript".
- Restaurar (borrar): <INPUT TYPE="RESET">. Atributos:
  - NAME="nombre".
  - VALUE="etiqueta".
  - ONCLICK="códigoScript".
- Botón de envío: <INPUT TYPE="SUBMIT">. Atributos:
  - NAME="nombre".
  - VALUE="etiqueta".
- Línea de texto: <INPUT TYPE="TEXT">. Atributos:
  - NAME="nombre".
  - VALUE="texto".
  - MAXLENGTH="máximoNúmeroCaracteres".
  - SIZE="longitudCampo".
  - ONBLUR="códigoScript".
  - ONCHANGE="códigoScript".
  - ONFOCUS="códigoScript".
  - ONSELECT="códigoScript".
- Lista de selección: <SELECT> ... </SELECT>. Atributos:
  - NAME="nombre".

- MULTIPLE.
- SIZE="longitudCampo".
- ONBLUR="códigoScript".
- ONCHANGE="códigoScript".
- ONCLICK="códigoScript".
- ONFOCUS="códigoScript".

▪ Opción en una lista de selección: <OPTION> ... </OPTION>. Atributos:

- VALUE="valor".
- SELECTED.

▪ Área de texto: <TEXTAREA> ... </TEXTAREA>. Atributos:

- NAME="nombre".
- COLS="columnas".
- ROWS="filas".
- WRAP="OFF" | "HARD" | "SOFT".
- ONBLUR="códigoScript".
- ONCHANGE="códigoScript".
- ONFOCUS="códigoScript".
- ONSELECT="códigoScript".

▪ Generador de clave: <KEYGEN>. Atributos:

- NAME="nombre".
- CHALLENGE="desafío".

▪ Elemento de búsqueda‹ 41 ›: <ISINDEX>. Atributo:

- PROMPT="texto".

A.11. Etiquetas de marcos

En esta sección se muestran las etiquetas que permiten crear marcos y conjuntos de marcos. Un marco es una región de una ventana que actúa como una ventana ella misma.

- Región de una ventana (marco): <FRAME>. Atributos:
  - BORDERCOLOR="color".
  - FRAMEBORDER="YES" | "NO".
  - MARGINHEIGHT="alturaMargen".
  - MARGINWIDTH="anchuraMargen".
  - NAME="nombreMarco".
  - NORESIZE.
  - SCROLLING="YES" | "NO" | "AUTO".
  - SRC="URL".
- Conjunto de marcos: <FRAMESET> ... </FRAMESET>. Atributos:
  - COLS="listaAnchurasColumnas".
  - ROWS="listaAlturasFilas".
  - BORDER="anchura".
  - BORDERCOLOR="color".
  - FRAMEBORDER="YES" | "NO".
  - ONBLUR="códigoScript".
  - ONFOCUS="códigoScript".
  - ONLOAD="códigoScript".
  - ONUNLOAD="códigoScript".
- Texto alternativo a los marcos: <NOFRAMES> ... </NOFRAMES>.

A.12. Etiquetas de situación de contenidos

En esta sección se encuentran las etiquetas que permiten definir la posición de los distintos elementos en una página. Estas etiquetas se emplean en **DHTML**.

- Definición de una capa: <LAYER> ... </LAYER>. Atributos:
  - ID="nombreCapa".
  - LEFT="posición".
  - TOP="posición".

- PAGEX="páginaX".
- PAGEY="páginaY".
- SRC="URL".
- Z-INDEX="número".
- ABOVE="nombreCapa".
- BELOW="nombreCapa".
- WIDTH="anchura".
- HEIGHT="altura".
- CLIP="número,número,número,número".
- VISIBILITY="visibilidad".
- BGCOLOR="color".
- BACKGROUND="URL".
- ONBLUR="códigoScript".
- ONFOCUS="códigoScript".
- ONLOAD="códigoScript".
- ONMOUSEOVER="códigoScript".
- ONMOUSEOUT="códigoScript".

▪ Capa posicionada de forma relativa: <ILAYER> ... </ILAYER>. Atributos:

- ID="nombreCapa".
- LEFT="posición".
- TOP="posición".
- PAGEX="páginaX".
- PAGEY="páginaY".
- SRC="URL".
- Z-INDEX="número".
- ABOVE="nombreCapa".
- BELOW="nombreCapa".
- WIDTH="anchura".
- HEIGHT="altura".
- CLIP="número,número,número,número".
- VISIBILITY="visibilidad".

- BGCOLOR="color".
- BACKGROUND="URL".
- ONBLUR="códigoScript".
- ONFOCUS="códigoScript".
- ONLOAD="códigoScript".
- ONMOUSEOVER="códigoScript".
- ONMOUSEOUT="códigoScript".

- Texto alternativo a las capas: <NOLAYER> ... </NOLAYER>.

A.13. Etiquetas de script

En esta sección se encuentran las etiquetas que permiten incluir código script en una página HTML.

- Código *JavaScript* en el cliente‹ 42 ›: <SCRIPT> ... </SCRIPT>. Atributos:
  - LANGUAGE="nombreLenguajeScript".
  - SRC="localizacionURL".
- Texto alternativo al código *JavaScript*: <NOSCRIPT> ... </NOSCRIPT>.
- Código en el servidor: <SERVER> ... </SERVER>.

A.14. Etiquetas de applets y plug-ins

Esta sección muestra las etiquetas que se emplean para incluir *applets* y objetos que emplean *plug-ins*.

- *Applet Java*: <APPLET> ... </APPLET>. Atributos:
  - NAME="nombre".
  - CODE="nombreFicheroClass".
  - CODEBASE="directorioFicheroClass".
  - ARCHIVE="archivo".

- ALT="textoAlternativo".
- ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
- HEIGHT="altura".
- WIDTH="anchura".
- HSPACE="margenHorizontal".
- VSPACE="margenVertical".
- MAYSCRIPT.

■ Parámetro para un *applet*: <PARAM>. Atributos:

- NAME="nombre".
- VALUE="valor".

■ *Plug-in* incrustado: <EMBED> ... </EMBED>. Atributos:

- NAME="nombre".
- SRC="URL".
- TYPE="tipoMIME".
- PLUGINSPAGE="URLinstrucciones".
- PLUGINURL="URLplugin".
- ALIGN="LEFT" | "RIGHT" | "TOP" | "BOTTOM".
- BORDER="anchura".
- FRAMEBORDER="YES" | "NO".
- HEIGHT="altura".
- WIDTH="anchura".
- UNITS="unidades".
- HIDDEN="TRUE" | "FALSE".
- HSPACE="margenHorizontal".
- VSPACE="margenVertical".
- PALETTE="FOREGROUND" | "BACKGROUND".

■ Texto alternativo para objetos incrustados: <NOEMBED> ... </NOEMBED>.

■ Objeto incrustado: <OBJECT>. Atributos:

- CLASSID="ficheroClase".
- DATA="URL".

- CODEBASE="directorioFicheroClase".
- TYPE="tipoMIME".
- ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
- HEIGHT="altura".
- WIDTH="anchura".
- ID="nombre".

A.15. Etiquetas de ajuste del texto

Las etiquetas de esta sección permiten ajustar la posición del texto.

- Salto de línea:
. Atributo:
  - CLEAR="ALL" | "LEFT" | "RIGHT".
- Centrado: <CENTER> ... </CENTER>.
- Línea horizontal: <HR>. Atributos:
  - ALIGN="CENTER" | "LEFT" | "RIGHT".
  - NOSHADE.
  - SIZE="grosor.
  - WIDTH="anchura".
- Múltiples columnas: <MULTICOL> ... </MULTICOL>. Atributos:
  - COLS="númeroColumnas".
  - GUTTER="separaciónColumnas".
  - WIDTH="anchuraColumnas".
- No salto de línea: <NOBR> ... </NOBR>.
- Espacio extra: <SPACER>. Atributos:
  - TYPE="HORIZONTAL" | "VERTICAL" | "BLOCK".
  - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
  - HEIGHT="altura".

- WIDTH="anchura".
- SIZE="tamaño".

- Intervalo de contenido: <SPAN> ... </SPAN>.
- Posible salto de línea: <WBR>.

A.16. Atributos universales

Los siguientes atributos se pueden emplear con prácticamente todas las etiquetas situadas en el cuerpo de una página <BODY> ... </BODY>.

- Estilo de la clase: CLASS="claseEstilo".
- Idioma: LANG="ISO".
- Nombre de lugar o de estilo: ID="nombreLugarOEstilo".
- Estilo: STYLE="estilo".

Apéndice B

Colores en HTML

El lenguaje HTML posee varias etiquetas con atributos que indican un color. Por ejemplo, la etiqueta <BODY> tiene el atributo BGCOLOR que permite indicar el color de fondo de una página y la etiqueta <FONT> posee el atributo COLOR para cambiar el color del texto. En este apéndice se explica como trabajar con los colores en HTML.

Índice General

B.1. Como trabajar con las componentes RGB

Las componentes **RGB** permiten expresar cualquier color mediante la combinación de los tres colores básicos (primarios) rojo, verde y azul. Cada componente expresa la intensidad del color básico en la combinación. Así, por ejemplo, el color negro es el resultado de combinar los tres colores básicos con una intensidad nula (0), mientras que el blanco es el resultado de combinar los tres colores con una intensidad máxima (255 cada componente). Este sistema de codificación de los colores permite 16.777.216 colores (256 x 256 x 256 combinaciones posibles).

Cuando se trabaja con los colores en **HTML**, las tres componentes se tienen que expresar en hexadecimal. Como este sistema de numeración es un poco "engorroso", a continuación mostramos como se pueden convertir las componentes **RGB** de decimal a hexadecimal mediante las herramientas que posee Microsoft Windows.

B.1.1. Obtener las componentes del color deseado en decimal

Mediante cualquier programa que permita seleccionar colores, podemos elegir el color que queremos emplear en una página **HTML**. Por ejemplo, en el programa Microsoft Paint‹ 43 ›, que viene con los sistemas operativos de Microsoft y que se puede encontrar a través de **Inicio** → **Programas** → **Accesorios**, si seleccionamos en el menú **Colores** la opción **Modificar colores. . .** aparece la ventana de la Figura B.1. Si en esta ventana se pulsa el botón **Definir colores personalizados** >>, la ventana anterior se amplia y aparece la ventana de la Figura B.2.



Figura B.1: Ventana para modificar colores en Microsoft Paint

En esta ventana se puede elegir de la paleta de colores un color y para el color elegido se puede seleccionar su brillo. En las casillas **Rojo**, **Verde** y **Azul** aparecen las correspondientes componentes en decimal (los valores de cada casilla varían desde 0 hasta 255). Una vez que se tienen las componentes del color deseado, el siguiente paso es convertirlas a hexadecimal.

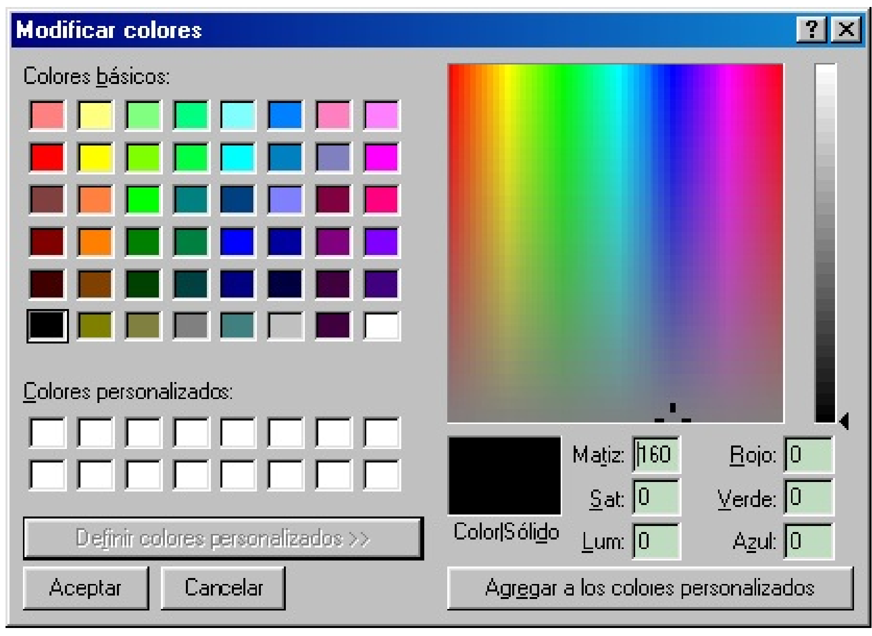


Figura B.2: Ventana para definir colores personalizados en Microsoft Paint

B.1.2. Transformar las componentes de decimal a hexadecimal

Para pasar un número del sistema decimal al hexadecimal, se tiene que dividir el número entre 16. Si el cociente es mayor que 15, se divide el cociente entre 16 y así sucesivamente hasta lograr un cociente menor que 16. El número en hexadecimal escrito de izquierda a derecha se compone del último cociente y los sucesivos restos obtenidos, desde el último hasta el primero, pero si el último cociente o los restos tienen dos cifras, se tienen que transformar según las equivalencias del Cuadro B.1.

Cociente o resto	Cifra hexadecimal
10	A
11	B
12	C
13	D
14	E
15	F

Cuadro B.1: Equivalencias para pasar del sistema decimal al hexadecimal

Por ejemplo, el número 241 en decimal equivale a F1 en hexadecimal, ya que el

primer (y único) resto que se obtiene es 1 y el último (y único) cociente 15, que equivale a F.

Como el método anterior es tedioso y propenso a errores, se puede realizar la conversión automáticamente mediante el ordenador. Para ello, se puede emplear el programa Calculadora que se incluye en los sistemas operativos de MICROSOFT y que se encuentra otra vez en **Inicio** → **Programas** → **Accesorios**. El programa Calculadora posee dos modos de visualización: estándar y científica. Para realizar la conversión, tiene que estar en el modo calculadora científica, que se selecciona a través del menú **Ver**.

Tal como se ve en la Figura B.3, el programa Calculadora tiene cuatro botones de radio marcados con las etiquetas **Hex** (hexadecimal), **Dec** (decimal), **Oct** (octal) y **Bin** (binario), que permiten convertir un número a los distintos sistemas de numeración. Para pasar de decimal a hexadecimal, simplemente hay que escribir el número cuando la calculadora se encuentra en el sistema **Dec** y pulsar el botón **Hex** para que aparezca en pantalla el número convertido a hexadecimal.

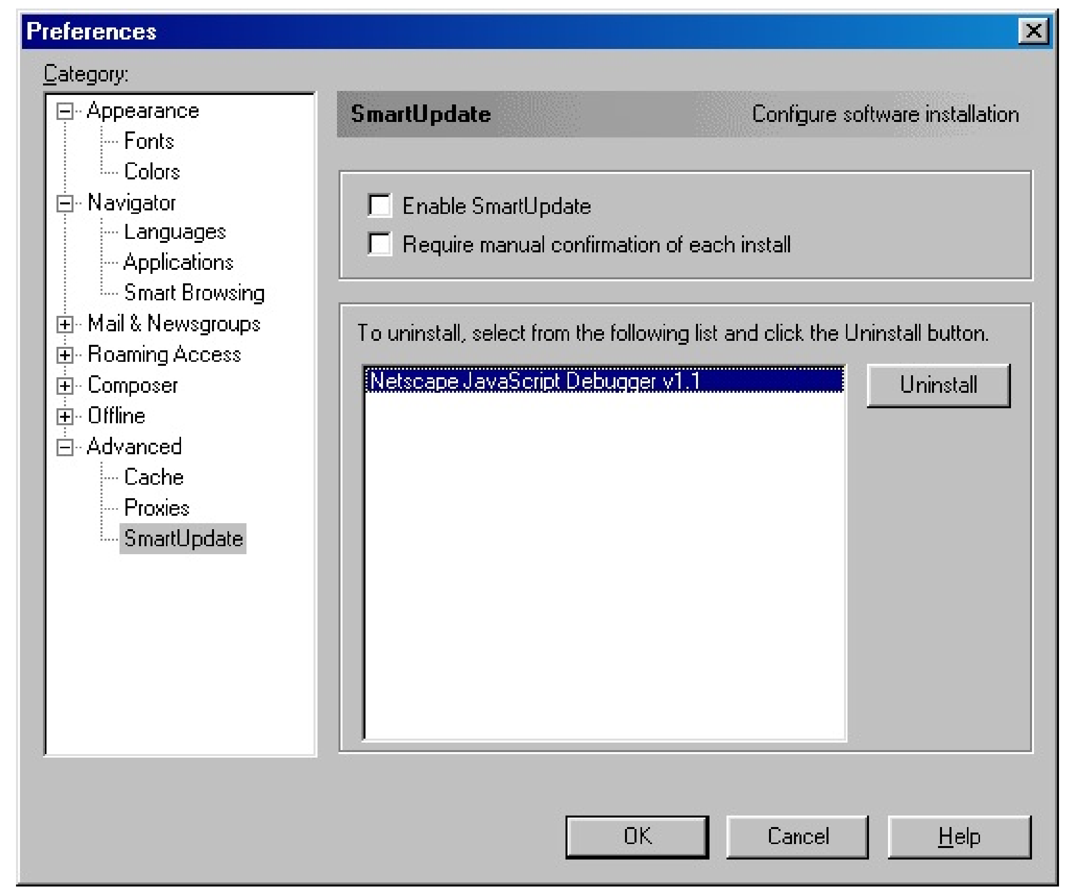


Figura B.3: Calculadora en modo científico

B.2. Tabla de colores

En **HTML** existe una serie de colores predefinidos a los que se les ha asignado un nombre en inglés. La lista de nombres contiene cientos de colores, por lo que aquí solo incluimos el Cuadro B.2 con algunos de ellos, junto con su valor en **RGB**.

Nombre color	**Valor RGB**
aqua	#00FFFF
black	#000000
blue	#0000FF
fuchsia	#FF00FF
gray	#808080
green	#008000
lime	#00FF00
maroon	#800000
navy	#000080
olive	#808000
purple	#800080
red	#FF0000
silver	#C0C0C0
teal	#008080
yellow	#FFFF00
white	#FFFFFF

Cuadro B.2: Nombres de algunos colores en HTML

Apéndice C

Depuración de errores de JavaScript

Como los lenguajes de script que se emplean en las páginas web son interpretados, la depuración de errores es difícil (al no existir la fase de compilación, no se detectan los errores sintácticos o semánticos). En este apéndice se explica como se puede depurar el código en cualquier navegador. Además, se comentan algunas herramientas específicas que poseen los navegadores Microsoft Internet Explorer y Netscape Communicator.

Índice General

C.1. Introducción

La depuración de errores de *JavaScript* suele estar desactivada en los navegadores, ya que sólo es útil para aquellos programadores que quieran verificar su código. Vamos a ver como podemos depurar código *JavaScript* de forma general en cualquier navegador y de forma específica mediante las características que nos ofrecen Netscape Communicator y Microsoft Internet Explorer.

C.2. Depuración en cualquier navegador

Se pueden emplear las ventanas de alerta de *JavaScript* para mostrar la información que deseemos durante la ejecución del código: valor de una variable, situación del

punto de ejecución del código, etc. Este método también permite detener momentáneamente la ejecución del código. Las ventajas que ofrece este sistema son:

1. La información que se muestra puede ser tan detallada como nosotros queramos.
2. Su uso no tiene efecto sobre el resto del código.
3. Se pueden emplear tantas ventanas de alerta como se quiera.
4. Se pueden colocar prácticamente en cualquier lugar del código.
5. Funciona con todos los navegadores que soporten código *JavaScript*.

Por ejemplo, si queremos comprobar en un punto del código el valor de una variable, sólo hay que incorporar una línea de código similar a la siguiente:

Ejemplo C.1

```
alert("El valor de la variable v es " + v);
```

C.3. Netscape Communicator

En versiones anteriores a la 4.06, cada vez que se producía un error de *JavaScript*, se mostraba una ventana de alerta con el error que se había producido. Este sistema era bastante engorroso, ya que si una página contenía múltiples errores, se mostraban múltiples ventanas que el usuario tenía que cerrar una a una.

A partir de la versión 4.06, que incluye *JavaScript* 1.3, se emplea la consola de *JavaScript* (Figura C.1). La consola sustituye a todas las ventanas de alerta: cada vez que se encuentra un error, se escribe un mensaje en la consola. La consola almacena todos los mensajes de error que se producen (Figura C.2). Además, la consola se puede dejar abierta o cerrar y abrir tantas veces como se quiera.

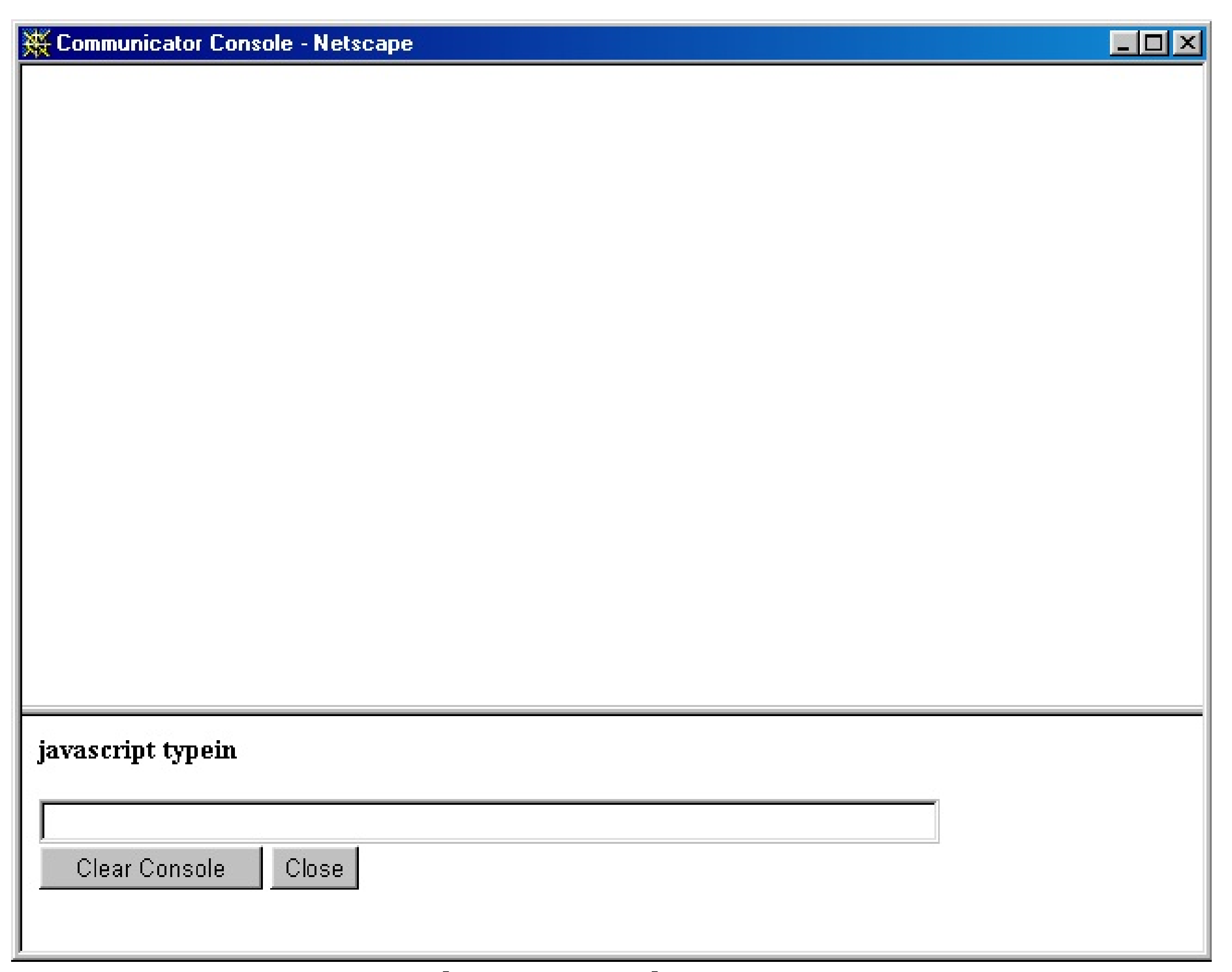


Figura C.1: Consola JavaScript de Netscape Communicator

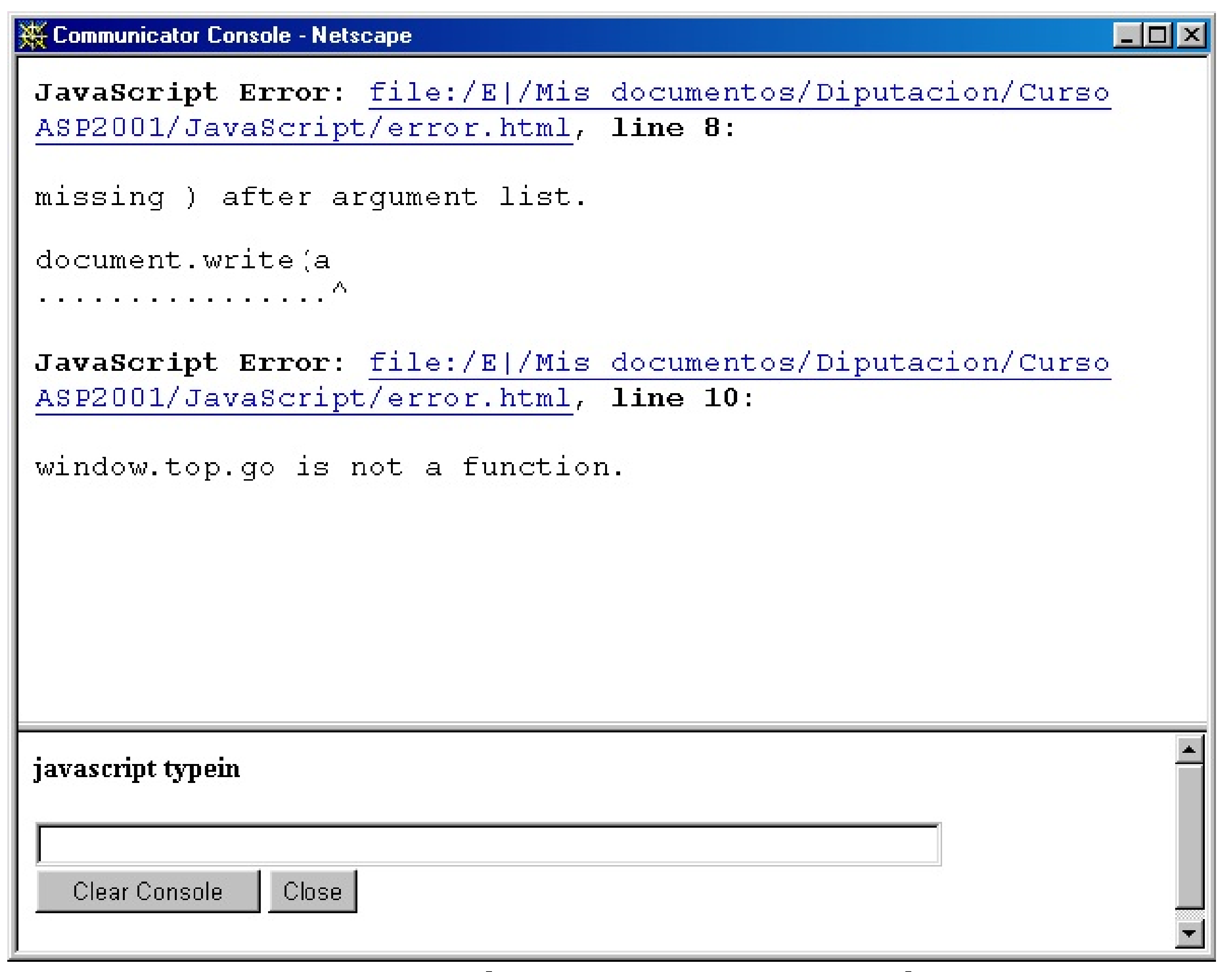


Figura C.2: Consola JavaScript con mensajes de error

Por defecto, la consola no se muestra: se encuentra oculta y no hay ninguna opción en los menús del navegador que permita mostrarla. Si se desea mostrar la consola, existen cuatro posibilidades:

1. Escribir javascript: en la barra de direcciones del navegador y pulsar la tecla Enter (Return).

2. Seleccionar en el menú **File** la opción **Open Page. . .** y escribir en el campo **URL** javascript:.

3. Incorporar en el código **HTML** de la página el siguiente enlace:

Ejemplo C.2

```
<A HREF="javascript:">Abrir consola JavaScript</A>}.
```

4. Modificar las preferencias del navegador para que se muestre automáticamente cada vez que se produzca un error. Esta opción es la mejor para los usuarios que estén desarrollando código *JavaScript*.

La consola *JavaScript* dispone de dos botones, tal como se ve en la Figura C.1:

- El botón **Clear Console** permite borrar todo los mensajes de error mostrados hasta el momento.
- El botón **Close** cierra la ventana de la consola.

Para cada error que se encuentra, en la consola *JavaScript* se muestra un mensaje con la **URL** del fichero que contiene el error, el número de línea y un comentario que describe el tipo de error. En algunos casos, se incluye también una parte de la línea que contiene el error y se marca el punto exacto donde se encuentra.

C.3.1. Modificar las preferencias

Para algunos propósitos, el tener que abrir de forma manual la consola *JavaScript* puede ser molesto. Por ejemplo, si estamos desarrollando una página, el tener que teclear javascript: cada vez que hay un error para abrir la consola y ver el mensaje de error puede ser tedioso.

Se puede especificar que automáticamente se abra la consola cuando se produzca un error de *JavaScript*. Para ello, hay que modificar el fichero de preferencias prefs.js, que se encuentra en el directorio personal de cada usuario de Netscape Communicator. Por defecto, suele ser C:\Archivos de programa\Netscape\Users\nombreUsuario.

Para modificar este fichero hay que seguir los siguientes pasos:

1. Asegurarse de que el navegador no está ejecutándose. El navegador puede sobrescribir los cambios que realicemos si se está ejecutando cuando editemos el fichero de preferencias.
2. Abrir el fichero de preferencias prefs.js. El contenido del fichero es similar a (sólo se muestran las primeras líneas):

Ejemplo C.3

```
// Netscape User Preferences
// This is a generated file! Do not edit.

user_pref("autoupdate.enabled", false);
user_pref("browser.bookmark_window_showwindow", 3);
user_pref("browser.cache.disk_cache_size", 20480);
user_pref("browser.cache.memory_cache_size", 2048);
user_pref("browser.download_directory", "D:\\TV\\");
```

3. Añadir una de las siguientes líneas al final del fichero:
   - Si se quiere que se abra automáticamente la consola cada vez que se produce un error de *JavaScript*:

Ejemplo C.4

```
user_pref("javascript.console.open_on_error", true);
```

   - Si se quiere que se abra una ventana de alerta cada vez que se produce un error de *JavaScript* (como el sistema empleado en las versiones anteriores):

Ejemplo C.5

```
user_pref("javascript.classic.error_alerts", true);
```

   Nota: aunque este sistema figura en la documentación de Netscape Communicator, se ha probado en la versión 4.7 y no funciona.

4. Grabar y cerrar el fichero prefs.js.

C.3.2. Evaluación de expresiones con la consola

La consola de *JavaScript* es un ventana compuesta de dos marcos (Figura C.1). El marco inferior contiene un cuadro de texto etiquetado javascript typein en el que se pueden escribir expresiones de una sola línea. Se puede emplear este cuadro de texto para asignar valores a variables, realizar operaciones matemáticas o verificar el resultado devuelto por los operadores de comparación.

Para evaluar una expresión simplemente hay que escribirla en el cuadro de texto y pulsar la tecla Enter (Return). El resultado se muestra en el marco superior. Por ejemplo, al evaluar las siguiente expresiones se obtienen los resultados citados en los comentarios y mostrados en la Figura C.3:

Ejemplo C.6

```
alert("Hola a todos"); // Muestra una ventana de alerta
5-2; // Muestra 3
Math.sqrt(49); // Muestra 7
var a=100; var b=45; // Crea dos variables
a-b; // Muestra 55
```

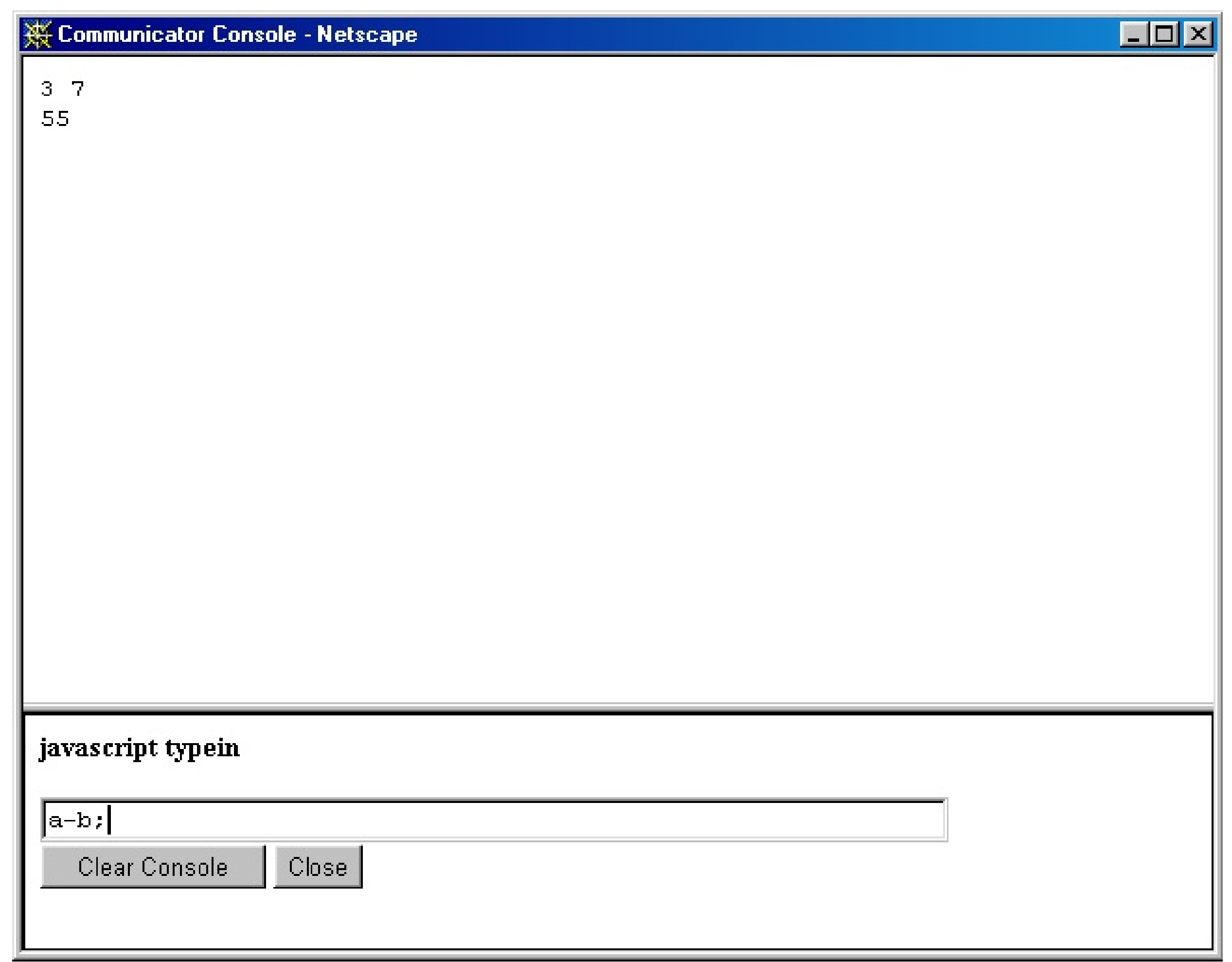


Figura C.3: Evaluación de expresiones

C.3.3. Netscape JavaScript Debugger

Existe una posibilidad más con el navegador de NETSCAPE: Netscape JavaScript Debugger (Figura C.4). Se trata de un *applet* (hecho en *Java*) que permite realizar una depuración avanzada del código *JavaScript*: visor de objetos (*object inspector*), puntos de parada (*breakpoints*), visores de variables (*watches*), ejecución paso a paso (*step running*), visor de la pila de ejecución (*call stack window*), etc.

Esta herramienta se puede descargar en las siguientes direcciones:

- http://developer.netscape.com/software/jsdebug.html.
- http://home.netscape.com/eng/Tools/JSDebugger/relnotes/relnotes11.html.

Para poderlo descargar e instalar, es necesario tener activada la opción **SmartUpdate**. Se accede a ella a través del menú **Edit** → **Preferences** → **Advanced** → **SmartUpdate** y activar la casilla **Enable SmartUpdate** (Figura C.5). El sistema

de descarga e instalación es automático.

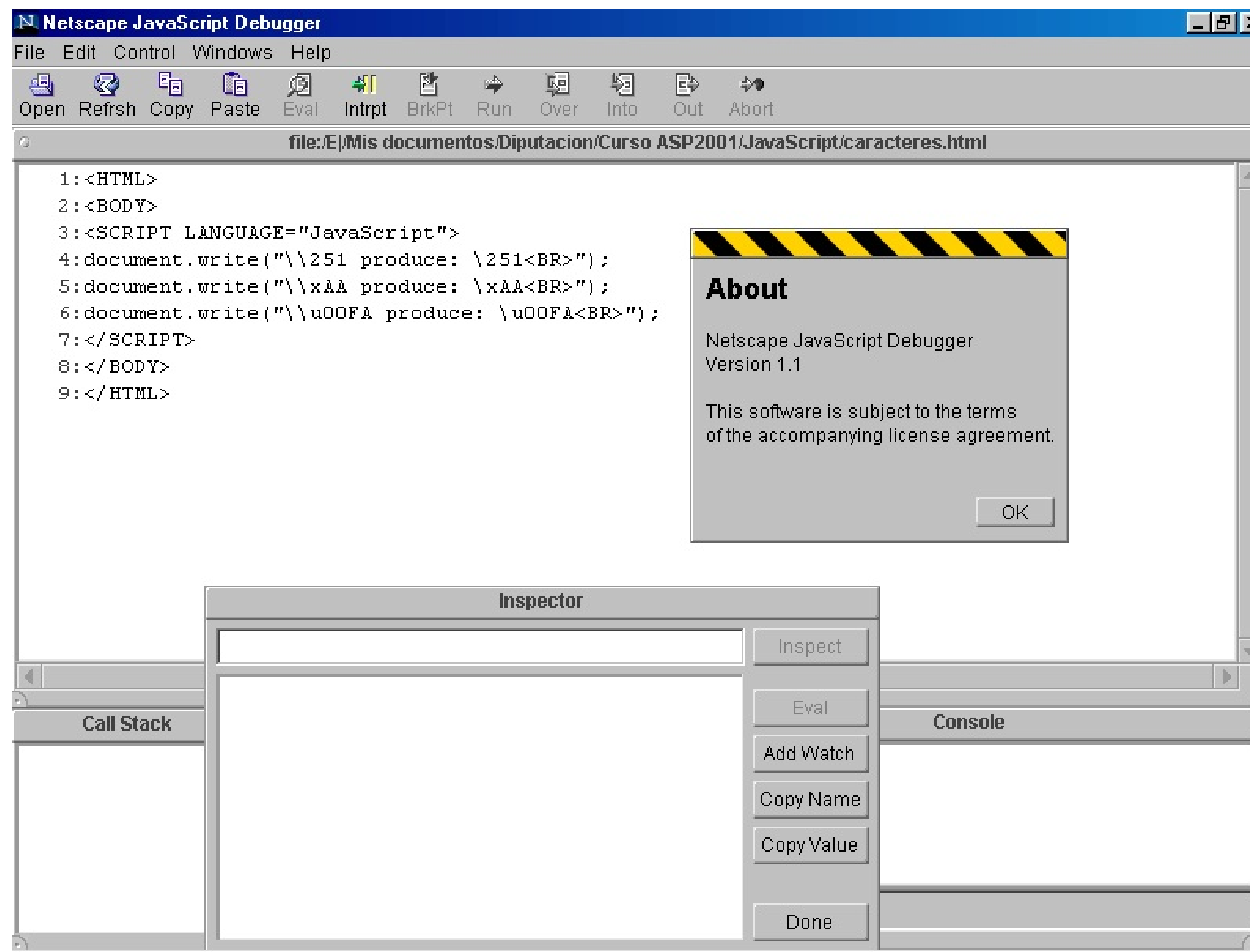


Figura C.4: Netscape JavaScript Debugger

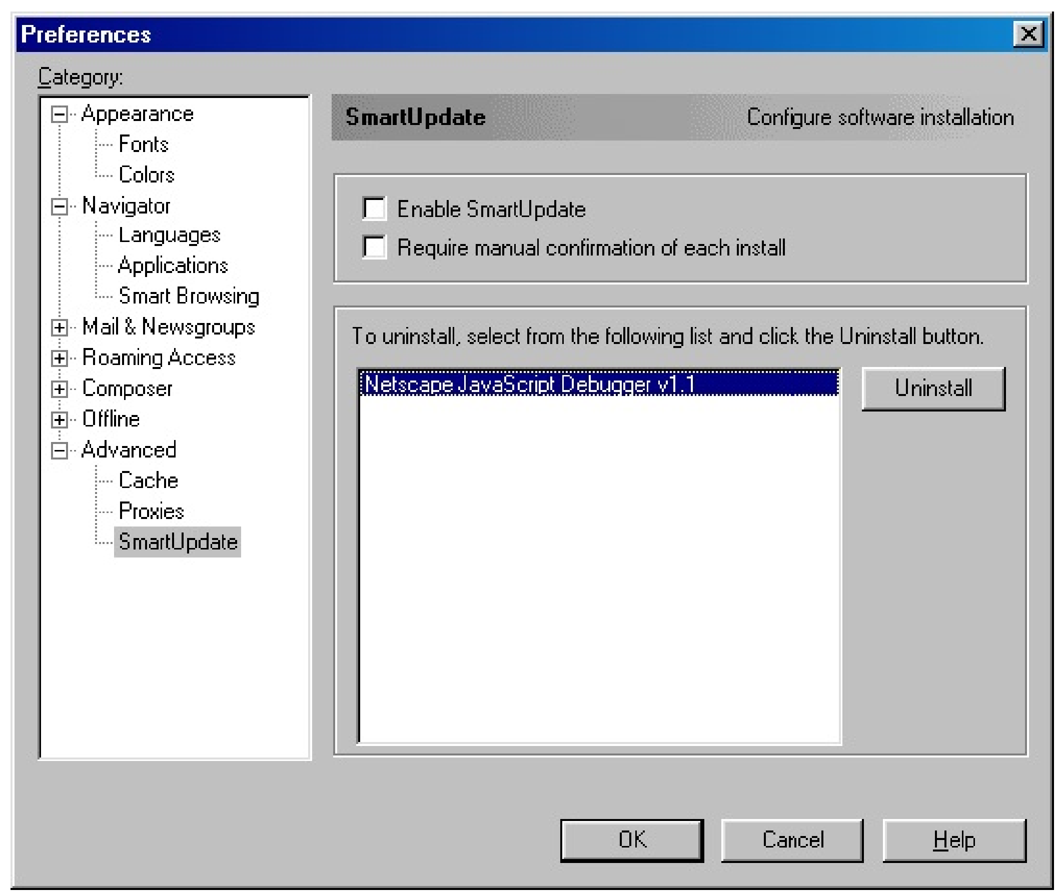


Figura C.5: SmartUpdate en Netscape Communicator

C.4. Microsoft Internet Explorer

En las versiones 5.0 o superiores de este navegador, se muestra automáticamente una ventana de alerta (Figura C.6) por cada uno de los errores de *JavaScript* que se encuentren en una página. Estas ventanas de alerta se pueden desactivar de dos formas:

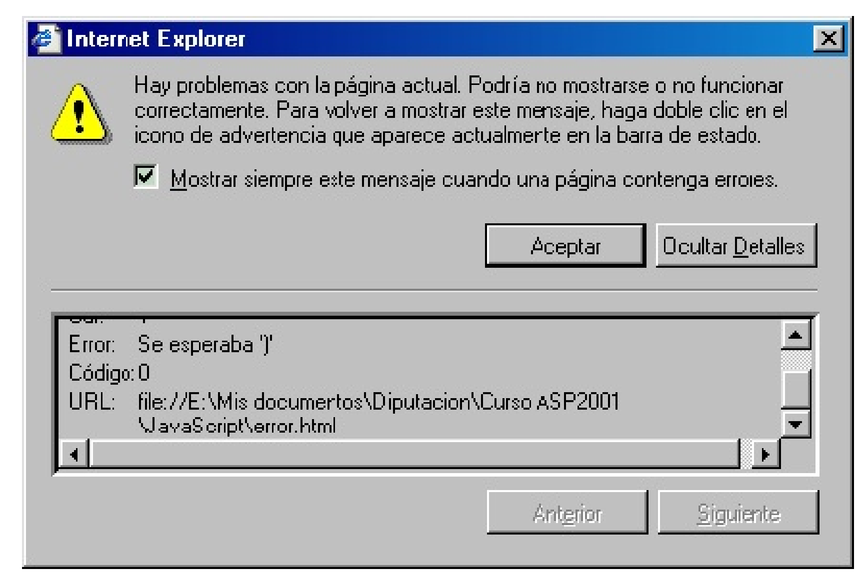


Figura C.6: Mensaje de alerta de Microsoft Internet Explorer

1. Desactivando la casilla **Mostrar siempre este mensaje cuando una página contenga errores** en la propia ventana de alerta. La próxima vez que se localice un error de *JavaScript*, no se mostrará la ventana de alerta.

2. Desactivando la opción **Mostrar una notificación sobre cada error de secuencia de comandos** en el menú **Herramientas** → **Opciones de Internet. . .** → **Avanzadas** (Figura C.8). A través de esta opción se puede volver a activar.

En cualquier caso (esté activado o desactivado), si se encuentra un error, en la barra de estado del navegador se muestra un mensaje (Figura C.7) cuando se localiza un error. Pulsando sobre el icono (triángulo amarillo), se abre la ventana de alerta.



Figura C.7: Mensaje de alerta en la barra de estado de Microsoft Internet Explorer

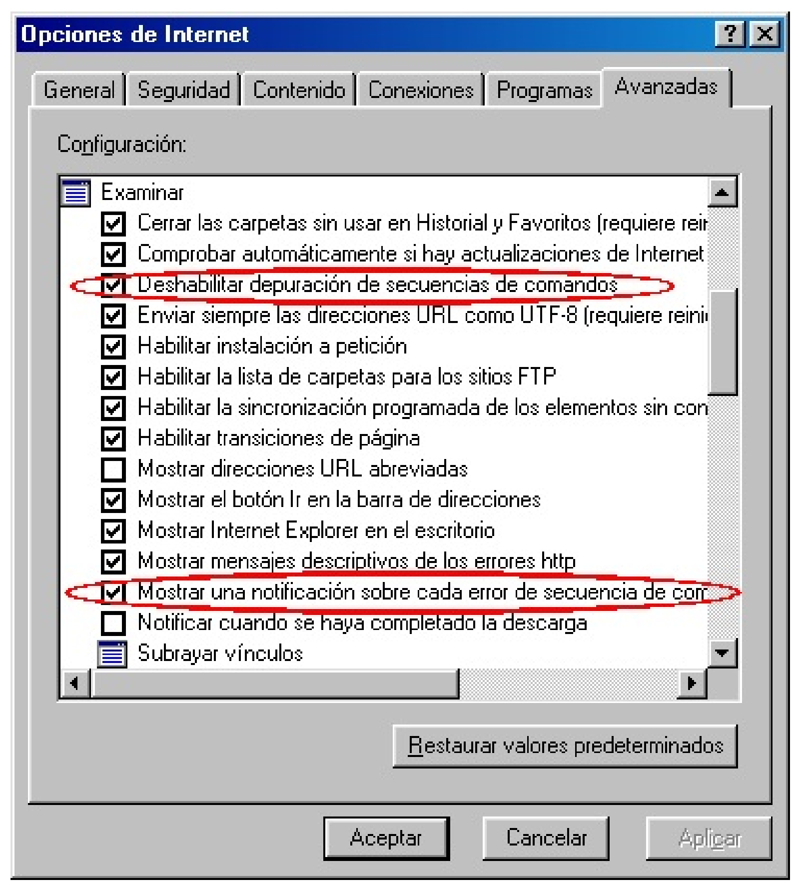


Figura C.8: Opciones de Microsoft Internet Explorer

En la ventana de alerta se muestra un mensaje con información sobre el error: línea, carácter, error, código y **URL** (fichero que contiene el error). Estos mensajes suelen ayudar bastante poco a localizar el error, ya que a veces no señalan el sitio exacto del error.

Si se tiene instalado algún entorno de programación de MICROSOFT, como *Visual Basic* o *Visual C++*, a Microsoft Internet Explorer se incorpora una opción de depuración de secuencia de comandos (código de *script*), tal como se ve en la Figura C.8 y Figura C.11. Esta opción permite depurar el código de *JavaScript* mediante el depurador empleado en los entornos de programación de MICROSOFT. El depurador (Figura C.12) permite realizar un depuración del código mucho más precisa, ya que incorpora inspección de variables, ejecución paso a paso, puntos de interrupción, visor de objetos, etc.

Para que funcione el depurador, tiene que estar desactivada la opción **Deshabilitar depuración de secuencias de comandos** en el menú **Herramientas** → **Opciones de Internet. . .** → **Avanzadas** (Figura C.8).

Cuando se tiene activada la opción de depurador de secuencias, las ventanas de alerta cambian a otras que muestran la línea donde se produce el error, el error producido y la posibilidad de abrir el depurador de secuencias para depurar el código de *JavaScript* (Figura C.9 y Figura C.10).



Figura C.9: Mensaje de error en Microsoft Internet Explorer

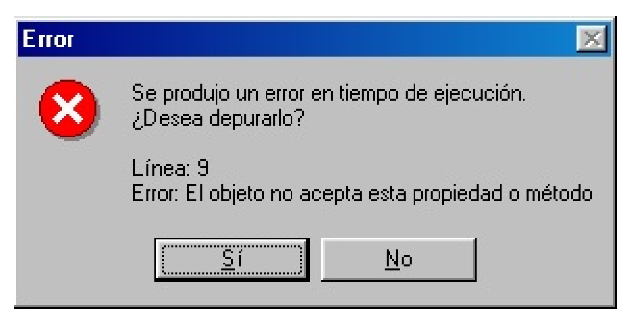


Figura C.10: Mensaje de error en Microsoft Internet Explorer



Figura C.11: Nuevas opciones de Microsoft Internet Explorer

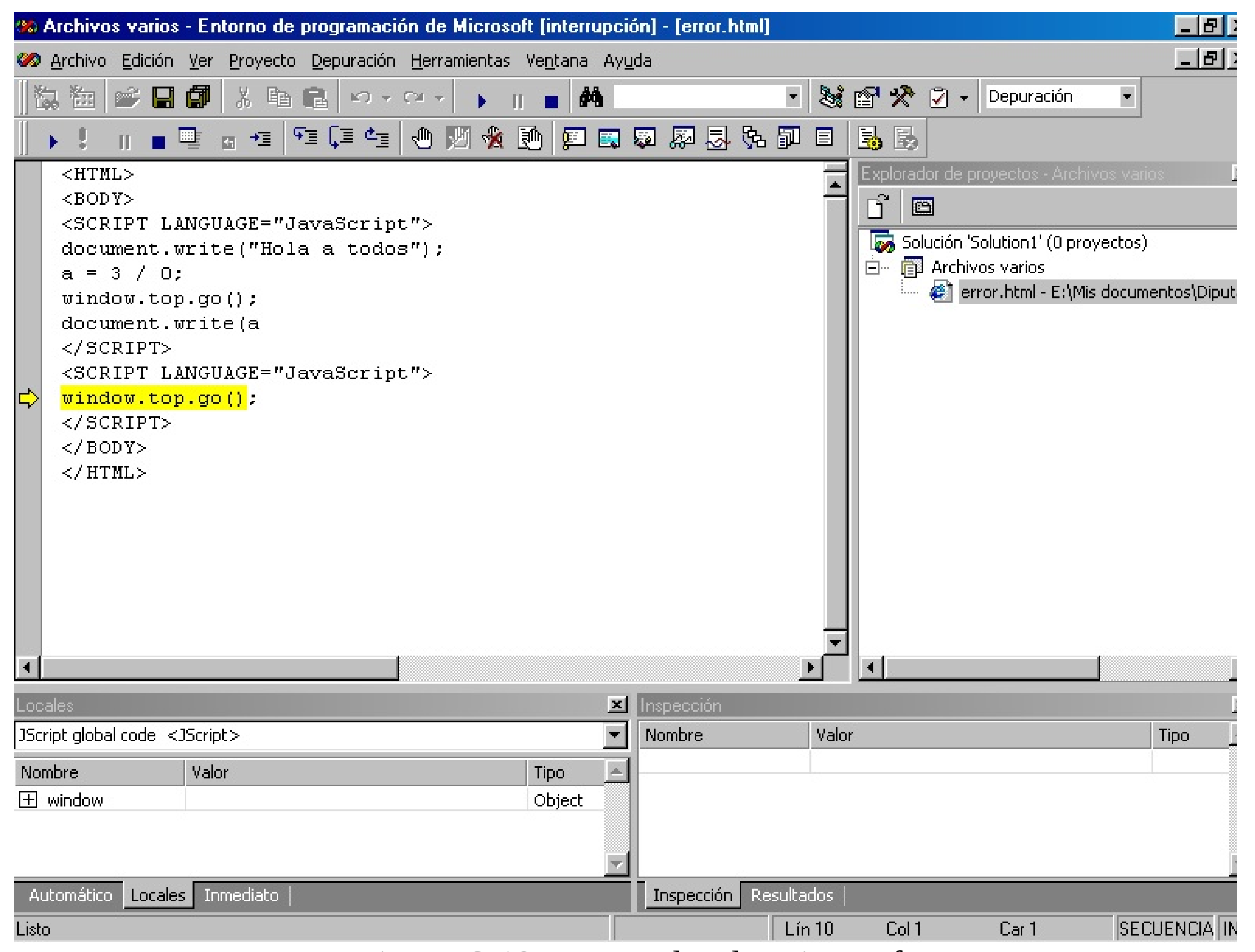


Figura C.12: Depurador de Microsoft

Bibliografía

[1] Danny Goodman. *Programación en JavaScript*. Vía@Internet, Anaya Multimedia, Madrid.

[2] Eduardo Parra Murga. *Diccionario de Internet*. PC World, IDG Communications, Madrid.

[3] H. M. Deitel, P. J. Deitel, T. R. Nieto. *Internet and World Wide Web. How to program*. Prentice Hall, New Jersey, 2000.

[4] Jesús Bobadilla Sancho. *Superutilidades para Webmasters*. Osborne McGraw-Hill, Madrid, 1999.

[5] José Manuel Alarcón. *Programación en JavaScript (actualizada hasta JavaScript 1.3 y JScript 5)*. Guías Prácticas, Anaya Multimedia, Madrid.

[6] Oscar González Moreno. *Programación en JavaScript*. Guías Prácticas, Anaya Multimedia, Madrid, 1998.

[7] Sergio Ríos Aguilar. *Lenguajes HTML, Java y CGI. El diseño de páginas Web para Internet a su alcance*. Abeto Editorial, Madrid, 1996.

NOTAS

‹ 1 › Otro tipo de arquitectura de red es *peer-to-peer* (entre pares o de igual a igual), en la que cada ordenador de la red posee responsabilidades equivalentes.

‹ 2 › *Application Program Interface*, interfaz de programación de aplicaciones.

‹ 3 › *Remote Procedure Call*, llamada a procedimiento remoto.

‹ 4 › *Open Database Connectivity*, conectividad de bases de datos abierta.

‹ 5 › En inglés se le suele denominar *browser*.

‹ 6 › Los clientes web también suelen actuar como clientes de transferencia de archivos (FTP), lectores de correo (SMTP y POP) y grupos de noticias (NNTP), etc.

‹ 7 › Un *plug-in* es un módulo de software que se instala como un añadido a un programa o sistema y que proporciona nuevas características o servicios al programa o sistema. En los navegadores, suelen permitir la reproducción de diferentes tipos de recursos de audio o vídeo.

‹ 8 › Tecnología de animación vectorial independiente de la plataforma, creada por MACROMEDIA INC.

‹ 9 › Pensemos, por ejemplo, en los ejecutivos que tienen que desplazarse entre distintos países, pero que necesitan acceder a las aplicaciones de su empresa.

‹ 10 › *Conseil Europeen pour le Recherche Nucleaire*.

‹ 11 › Tim Berners-Lee, Robert Cailliau, Jean-François Groff, Bernd Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, Vol. 1, No. 2, Meckler, Westport CT, primavera 1992.

‹ 12 › *Defense Advanced Research Projects Agency*.

‹ 13 › *Institut National de Recherche en Informatique et Automatique*.

‹ 14 › Algunas versiones, como **HTML+** o **HTML 3.0** nunca llegaron a estándar.

‹ 15 › Las páginas estáticas también pueden mostrar datos procedentes de una base de datos mediante la técnica *snap shot*. La información de la base de datos se convierte en **HTML** de forma manual, automáticamente cuando ocurre un suceso o a

una fecha y hora dadas (por ejemplo, todos los días a las tres de la mañana). Esta técnica es adecuada para catálogos, listas de precios, directorios telefónicos, etc., que no se modifican muy a menudo.

‹ 16 › Esta extensión existe debido a que en DOS y Microsoft Windows 3.x los ficheros sólo pueden tener una extensión de tres caracteres.

‹ 17 › Aunque los nombres de los archivos pueden tener mayúsculas y minúsculas, para acceder a un archivo no se tienen en cuenta. Por ello, en un mismo directorio no pueden existir dos archivos que sólo se diferencian porque algunos caracteres aparecen en mayúsculas en uno y en minúsculas en el otro.

‹ 18 › Los navegadores actuales son muy flexibles: si falta alguna etiqueta de fin no producen un error y muestran la página lo mejor posible. De todas formas, es recomendable ajustarse siempre a la sintaxis y no cometer errores.

‹ 19 › Información sobre la información. Los metadatos, por ejemplo, permiten indicar cómo, cuándo y quién ha recogido una información y como le ha dado formato.

‹ 20 › *Standard for ARPA Internet Text Messages*.

‹ 21 › En la documentación de Netscape Communicator no figura el valor JUSTIFY.

‹ 22 › Se puede cambiar con la etiqueta <BASEFONT SIZE="numero">.

‹ 23 › Las razones pueden ser varias: los ordenadores no tenían la suficiente potencia para manejar varias imágenes a la vez, el ancho de banda en la comunicaciones era menor o a nadie se le había ocurrido la idea de hacer un uso "intensivo y extensivo" de las imágenes.

‹ 24 › Compresión de la información en la que todos los datos iniciales se almacenan, por lo que la calidad de las imágenes no se ve afectada al recuperar las imágenes una vez comprimidas.

‹ 25 › El navegador conoce el tamaño de las imágenes antes de cargarlas, por lo que ya puede reservar el correspondiente espacio en el diseño de la página.

‹ 26 › Normalmente se utiliza POST, que indica que los datos se envíen por la entrada estándar. Si se utiliza GET, los datos se envían unidos a la **URL**.

‹ 27 › Un campo de contraseña es idéntico a un campo de texto, pero los caracteres se ocultan mediante asteriscos.

‹ 28 › Para seleccionar varias opciones se emplean la tecla **Control** para seleccionar de una en una y la tecla **Mays** para seleccionar un conjunto contiguo de opciones (se marca la primera y la última).

‹ 29 › Por ejemplo, para el lenguaje *JavaScript* podemos usar los identificadores *JavaScript*, *JavaScript1.1*, *JavaScript1.2* y *JavaScript1.3*.

‹ 30 › Tanto Netscape Communicator como Microsoft Internet Explorer, si no se indica un lenguaje con la etiqueta LANGUAGE, suponen que se trata de *JavaScript*.

‹ 31 › 3No se recomienda mezclar en una misma página distintos lenguajes de *script*: su mantenimiento es más difícil y su ejecución es más lenta, ya que se tiene que activar un intérprete distinto para cada lenguaje.

‹ 32 › *"Netscape and Sun announce Javascript, the open, cross-platform object scripting language for enterprise networks and the Internet"*, nota de prensa disponible en http://home.netscape.com/newsref/pr/newsrelease67.html.

‹ 33 › A partir de ahora, cuando hagamos referencia a C, se entiende que también se incluyen C++ y Java.

‹ 34 › Se permite el punto y la coma como separadores decimales.

‹ 35 › Cuando se quiera acceder a un objeto de otra ventana, habrá que indicar la ventana.

‹ 36 › Recordemos que se puede obviar.

‹ 37 › También se puede acceder a través de forms, un array donde cada posición representa un formulario.

‹ 38 › También puede emplearse en el cuerpo.

‹ 39 › También puede emplearse en el cuerpo.

‹ 40 › En la documentación de Netscape Communicator no figura el valor JUSTIFY.

‹ 41 › También puede emplearse en la cabecera.

‹ 42 › También puede emplearse en la cabecera.

‹ 43 › Se ha elegido este programa porque se encuentra en la mayoría de los ordenadores con sistema operativo Microsoft Windows. Existen programa de dibujo o retoque fotográfico, como Jasc Paint Shop Pro, que facilitan la tarea, ya que directamente muestran en hexadecimal las componentes **RGB** de un color.